Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Generování tetrahedronových sítí z rozptýlených dat

Pavel Maur
Informatika a výpočetní technika
Počítačová grafika a virtuální realita

Plzeň, 2000

Obsah

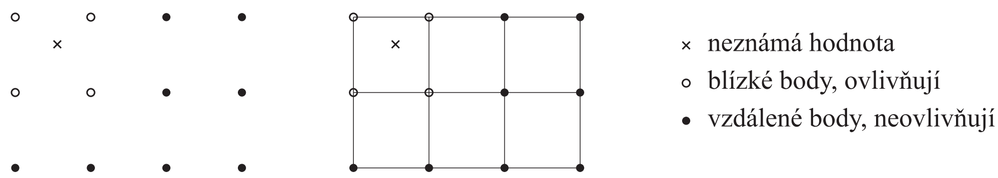
Kapitola 1

Úvod

1.1 Trojúhelníkové sítě

Představme si množinu bodů, jež jsou zadány svými souřadnicemi v prostoru. Uvažujme dále, že každý bod nese nějakou další informaci, např. hodnotu teploty naměřené v daném místě (tyto veličiny nemusí být jen skalární, mohou to být i vektory – např. směr proudění v daném místě apod.). Pokud chceme výše uvedená data zobrazit, musíme získat hodnoty měřené veličiny (teploty) i v bodech, kde měření neproběhlo. Znamená to, že v každém takovém bodu je nutno odvodit hodnotu na základě několika sousedních naměřených hodnot. Dá se totiž předpokládat, že místa v okolí mají na výslednou hodnotu v neměřeném místě podstatný vliv, na rozdíl od míst, která jsou mu vzdálena. Které body měření však považovat za sousední?

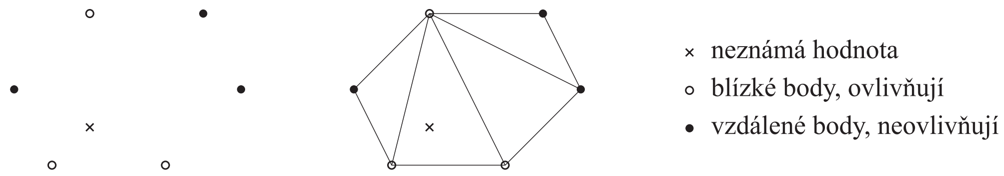
Pokud je síť měřených bodů rozmístěna pravidelně např. ve 2D, viz obrázek 1.1, existuje na otázku sousednosti jednoduchá odpověď. Můžeme si představit, že síť bodů vytváří obdélníky. Chceme-li zjistit měřenou hodnotu v místě mimo vrcholy sítě, najdeme ten z obdélníků, jemuž daný bod náleží. Vrcholy tohoto obdélníku jsou těmi hledanými sousedy, z nichž bude nová hodnota odvozena.



Obrázek 1.1: Pravidelná síť bodů

Jsou-li však body sítě rozmístěny nepravidelně, není již odpověď na otázku sousednosti tak zřejmá. To je patrné z obrázku 1.2. Proto se nad danou množinou bodů vytváří síť trojúhelníků (ve třírozměrném případě pak síť tetrahedronů). Každý bod, v němž neproběhlo měření, náleží do jednoho trojúhelníku (tetrahedronu) a hodnota měřené veličiny se opět určuje z hodnot ve vrcholech tohoto tělesa.

Tetrahedronové sítě jsou tedy prostředkem pro organizaci rozptýlených dat ve třírozměrném prostoru. Jak zjistíme dále, úloha vytvoření sítě není jednoznačná – nad danou množinou bodů lze vytvořit více různých sítí. Ne každá síť však musí být vhodná pro naše záměry. Pro výše uvedený příklad interpolace, např. pro vizualizaci, budeme požadovat tetrahedrony s vyrovnanými délkami hran, aby nedocházelo k negativnímu ovlivňování výsledku ze vzdálených naměřených míst.



Obrázek 1.2: Nepravidelná síť bodů

1.2 Pojmy a terminologie

Složitost algoritmu Rozlišujeme *časovou* a *paměťovou* (prostorovou) složitost algoritmu. *Časová složitost* značí počet elementárních operací, které je nutno vykonat během provádění algoritmu. Udává se v závislosti na počtu vstupních dat N, označuje se $\mathcal{O}(f(N))$. Algoritmus, který pro každou z N vstupních hodnot provede konstantní počet operací (např. výpis souřadnic daného vrcholu), má časovou složitost $\mathcal{O}(N)$. Závisí tedy na množství vstupních dat lineárně. To znamená, že dvojnásobné množství dat bude zpracováváno dvakrát déle. Pokud by časová složitost byla např. $\mathcal{O}(N^2)$, tj. kvadratická, trvalo by zpracování dvojnásobného počtu dat čtyřikrát déle.

Podobně můžeme vyjádřit *paměťovou složitost* algoritmu jako počet elementárních míst v paměti, které program zabere během svého chodu pro uložení dat.

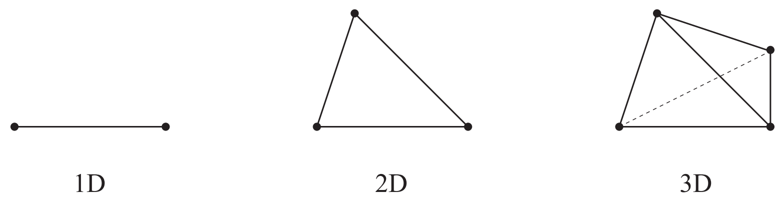
Nejhorší případ Složitost některých algoritmů je závislá na charakteru vstupních dat (např. na rozmístění bodů v prostoru – body můžou tvořit shluky nebo být naopak rozmístěny pravidelně). O takových algoritmech říkáme, že jsou závislé na výstupu. *Nejhorší případ* nás informuje o nejhorší možné složitosti algoritmu, která nastane, pokud jsou vstupní data „špatná“ pro daný algoritmus.

Očekávaný případ Skutečná data jsou málokdy tak „špatná“, aby docházelo k nejhoršímu případu. *Očekávaný případ* nám proto říká, jak se bude algoritmus chovat, pokud jsou vstupní data rozmístěna typicky pro danou aplikaci. Složitost očekávanáného případu je tak lepší, nejhůře stejná, jako složitost nejhoršího případu (např. tetrahedronová síť může být za jistých okolností vytvořena s očekávanou složitostí $\mathcal{O}(N^{4/3})$, zatímco nejhorší složitost je $\mathcal{O}(N^2)$ – viz [Joe91]).

Konvexní obálka (konvexní obal) množiny bodů je nejmenší konvexní objekt, který obsahuje všechny body dané množiny. Objekt je konvexní právě tehdy, když úsečka spojující libovolné dva body daného objektu leží celá uvnitř tohoto objektu. Viz obrázek 1.3. *Konvexní obálka* množiny bodů M se značí $CH(M)$ (z angl. *Convex Hull*).

Obrázek 1.3: Množina bodů a její konvexní obálka

Simplex je takový útvar, který má v prostoru dimenze D $D+1$ vrcholů a sám tento útvar má dimenzi D. Na obrázku 1.4 jsou zobrazeny *simplexy* pro jedno, dvou a třídimenzionální prostory – úsečka, trojúhelník a čtyřstěn. **Čtyřstěn** je někdy také označován jako **tetrahedron**.



Obrázek 1.4: Simplexy v prostorech různých dimenzí – úsečka, trojúhelník, čtyřstěn

Triangulace V literatuře se pojmem *triangulace* (triangulation) označuje proces vytváření 2D trojúhelníkových sítí. Někdy se také objevuje termín *triangularizace*. Pro konstrukci sítí ve 3D prostoru se užívá termínů *tetrahedronizace* (tetrahedronization), *tetrahedralizace* (tetrahedralization), nebo jen jednoduše *3D triangulace*.

1.3 Tetrahedronizace v této práci

Jak již bylo uvedeno dříve, základem pro řešení vizualizačních, interpolačních a jiných problémů nad nepravidelně rozmístěnými body ve třírozměrném prostoru je vytvoření sítě tetrahedronů nad danými body. Postup vytvoření takové sítě však není jednoznačný – nad danou množinou bodů lze sestrojit více sítí, které se od sebe navzájem liší např. počtem vytvořených tetrahedronů, jejich tvary, apod. Někdy jsou dokonce do základní množiny bodů vkládány další, tzv. Steinerovy body, a to kvůli dosažení lepších výsledků (vhodnějších tvarů tetrahedronů) – tato technika není v této práci užita.

Cílem této práce je vytvořit tetrahedronové sítě několika různými postupy a porovnat vzájemně jak kvalitu výsledných sítí, tak vlastnosti použitých algoritmů. Za tímto účelem budou vybrána kritéria hodnotící tvary tetrahedronů, z jejichž výpočtu pak bude možno ohodnotit kvalitu celé sítě. Také kritéria, zejména pak jejich vhodnost a vypovídací schopnosti, budou rovněž předmětem našeho zkoumání. Takto získané poznatky pak mohou sloužit pro další úkoly nad vytvořenými sítěmi, např. již zmiňované interpolace či hledání izoploch (množin bodů, kde měřená veličina nabývá nějakou konkrétní konstantní hodnotu) – tyto úkoly však již překračují rámec práce a nejsou zde uváděny.

Jak je jistě patrné, jsou tetrahedronové sítě ve třírozměrném prostoru zobecněním plošných triangulací z prostoru dvourozměrného. Ve dvourozměrném případu je dokázáno, že trojúhelníkovou síť lze vytvořit optimálně v čase $\mathcal{O}(N \log N)$, přičemž počet trojúhelníků je lineárně závislý na počtu bodů – paměťová složitost je $\mathcal{O}(N)$. Lze dokonce dokázat, že pro daný počet bodů N závisí počet vytvořených trojúhelníků pouze na množství bodů v konvexní obálce. Ve třírozměrném případu však tato pravidla neplatí. Počet vytvořených tetrahedronů se pro tutéž množinu může pohybovat od $\mathcal{O}(N)$ až do $\mathcal{O}(N^2)$ (blíže se této otázce věnuje část 2.2), což podstatným způsobem zvyšuje paměťovou složitost celého procesu, zvláště uvědomíme-li si, že již tak pracujeme s velkými objemy dat. Proto je nutné, aby použité algoritmy byly efektivní.

1.4 Organizace texu

Následující text je rozdělen na část teoretickou s popisem metod, které jistým způsobem přispěly k formování práce. Jsou to zejména *inkrementální algoritmus* vytváření tetrahedronové sítě, ve kterém je síť postupně budována přidáváním dalších bodů k již existující síti. Dále myšlenka *lokálních transformací*, podle které může být daná síť lokálně optimalizována vzhledem k nějakému kritériu prohazováním stěn sousedních tetrahedronů. Je zde také popsána alternativní metoda k lokálním transformacím – Watsonův algoritmus. Další uvedený postup buduje danou síť v očekávaném čase $\mathcal{O}(N)$. Nakonec jsou představena tvarová kritéria pro hodnocení tetrahedronů a uvedeny možné způsoby jejich použití pro porovnávání tetrahedronových sítí.

Část implementační nejprve popisuje vybraný způsob řešení, dále se věnuje užitým datovým strukturám a schematicky znázorňuje důležité partie programu – inkrementální postup vytvoření sítě, lokální transformace, postup určení jednotlivých typů konfigurací. Přehledově je zmíněna celková struktura programu a jeho začlenění do již existujícího celku, MVE.

Část věnující se výsledkům ilustruje časové poměry algoritmu v tabulkách a grafech, představuje a hodnotí dosažené výsledky.

V závěrečné části je celá práce ještě jednou stručně shrnuta. Je diskutováno splnění předkládaného zadání a zhodnoceny očekávané a dosažené výsledky.

Tato práce je součástí projektu VS 97155 Ministerstva školství České republiky.

Kapitola 2

Teoretická část

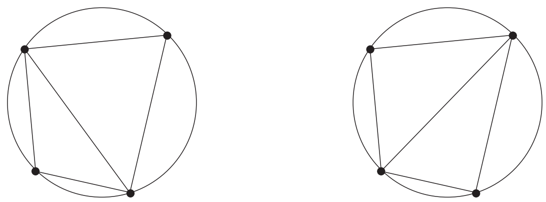
Jak jsme již uvedli, nemají konstrukční postupy sítí čtyřstěnů tak výhodné vlastnosti jako jejich analog ve 2D, plošné triangulace. Komplikace přináší zejména proměnný počet tetrahedronů v lineární až kvadratické závislosti na počtu bodů. Proto jsou časové složitosti užitých algoritmů často závislé na výstupu. Dosahuje se např. časové složitosti $\mathcal{O}(N \log N + K)$, kde K reprezentuje právě počet vzniklých tetrahedronů – pokud je $K \sim N$, získáme časovou složitost $\mathcal{O}(N \log N)$, pokud je $K \sim N^2$, výsledná časová složitost se zhorší na $\mathcal{O}(N^2)$. Jak si však ukážeme, může být očekávaná časová složitost některých algoritmů dokonce i $\mathcal{O}(N)$, v nejhorším případě však opět $\mathcal{O}(N^2)$.

Jak hodnotit vytvořené sítě? Bylo již řečeno, že např. pro interpolační účely je žádoucí, aby se na výpočtu hodnoty v neměřeném místě podíleli nejbližší sousedé. Pro tvar tetrahedronů to znamená, aby nebyly příliš „protáhlé“ a nezasahovaly tak do míst příliš vzdálených, ale aby se naopak blížily rovnostrannému případu. Pro ilustraci je uvedena tato situace na obrázku 2.1, pro přehlednost pouze ve 2D.

Obrázek 2.1: Ukázka možných konfigurací nad stejnými body. Vlevo tvary vhodnější pro interpolaci.

Jako kritéria hodnotící kvalitu tetrahedronů jsou používány různé geometrické veličiny, zejména poměry délek hran, rozměry koule opsané a vepsané tetrahedronu, vlastnosti týkající se úhlových poměrů. Pokud provádíme optimalizaci sítě vzhledem k některému z kritérií, jedná se většinou o lokální přístupy neboť pokus o nalezení globálního extrému vede na exponenciální časovou složitost (tj. vytvoření všech existujících triangulací nad danými body a výběr té „nejlepší“, což je pro větší počet bodů prakticky neřešitelný problém).

Výjimku potvrzující pravidlo tvoří kritérium *maximalizace minimálního úhlu*, kdy lokální optimalizací dosáhneme globálně optimální sítě. Tento postup se ve 2D nazývá *Delaunayova triangulace*, ve 3D *Delaunayova tetrahedronizace*. Je poměrně oblíben, neboť kritérium maximalizace minimálního úhlu v praxi znamená, že vytvořené simplexy jsou pokud možno co nejrovnostrannější, což – jak bylo uvedeno dříve – je ve většině případů žádoucí tvar. Praktické vytvoření Delaunayovy sítě se však žádným způsobem nezabývá měřením a porovnáváním úhlů, nýbrž používá ekvivalentní kritérium tzv. kružnice (koule) opsané. To říká: aby daný simplex splňoval Delaunayovo kritérium maximálního minimálního úhlu, nesmí uvnitř kružnice (koule) opsané simplexu ležet žádný jiný bod množiny. Problémy, které mohou nastat při vytváření Delaunayovy sítě, vycházejí z nejednoznačnosti takové sítě, pokud na kružnici (kouli) opsané leží více než tři (čtyři) body. Pak, jak ukazuje obrázek 2.2 pro 2D, je možno vytvořit více triangulací, přičemž každá z nich je správná.



Obrázek 2.2: Dva možné způsoby Delaunayovy triangulace, pokud více než tři body leží na kružnici

V následujících částech budou popsány základní přístupy, na nichž je tato práce založena. Nejprve se zmíníme o *inkrementálním vytvoření tetrahedronové sítě*. Jedná se o postup zobecněný z dvourozměrného prostoru, který je založen na postupném přidávání bodů do již vytvořené sítě. Zároveň bude objasněno, proč se počet vytvořených tetrahedronů může pohybovat v lineární až kvadratické závislosti na počtu bodů. Poté bude vysvětlen pojem *lokální transformace*, díky nimž je možno vytvořenou síť optimalizovat prohazováním stěn tetrahedronů. Lokální transformace budou představeny přímo na algoritmu 3D Delaunayovy triangulace. Jako alternativní postup k lokálním transformacím bude uveden Watsonův algoritmus. Dalším alternativním postupem, zajímavým pro změnu z časového hlediska, bude vytvoření 3D Delaunayovy triangulace s časovou složitostí $\mathcal{O}(N)$ – pouze jako ukázka možné urychlující techniky. Následovat bude popis tvarových kritérií pro hodnocení tetrahedronů.

2.1 Inkrementální algoritmus

Tento postup vznikl zobecněním z dvourozměrného prostoru (viz např. [Kol97]). Jeho nevýhodou je, že vytvořené tetrahedrony (trojúhelníky ve 2D) mají většinou protáhlý tvar, což se může negativně projevit např. při použití vytvořené sítě k interpolačním účelům. Výhodou naopak je – protože síť je vytvářena postupně – možnost optimalizace sítě po každém kroku vzhledem k libovolnému lokálnímu kritériu. Tím může být problém špatně tvarovaných simplexů odstraněn.

Mějme v prostoru množinu bodů, mezi nimiž chceme vytvořit tetrahedronovou síť. Pro začátek si tuto množinu představme jako jednoduchou – tedy takovou, kde všechny body jsou uzavřeny v jednom tetrahedronu (který tím zároveň tvoří konvexní obálku dané množiny bodů). Tento předpoklad je samozřejme pro obecně rozmístěné body velmi málo pravděpodobný, nám však poslouží k názornějšímu popisu celého postupu bez újmy na obecnosti. Nakonec si ukážeme, jak se bez něj obejít.

Máme tedy jeden tetrahedron a v něm uzavřeny všechny body. Inkrementální algoritmus lze popsat takto:

```
while (množina vstupních bodů není prázdná)
  vyber bod ze vstupní množiny
  lokalizuj tetrahedron, v němž se nalézá
  rozděl tetrahedron
  proveď případné další akce (např. lokální transformace)
```

Algoritmus se tedy skládá ze dvou klíčových částí: *vyhledání* tetrahedronu, ve kterém je bod obsažen, a *rozdělení* nalezeného tetrahedronu. Podívejme se nejprve na druhou z nich.

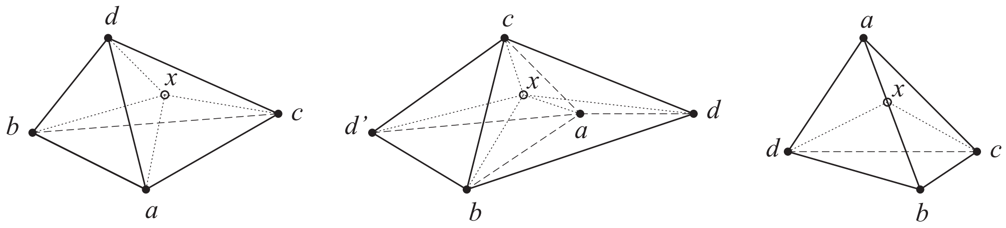
2.1.1 Rozdělení tetrahedronu

Představme si, že již máme nalezen tetrahedron, v němž se přidávaný bod nachází. Pokud vyloučíme možnost, kdy bod je totožný s některým vrcholem tetrahedronu, můžeme vzájemnou polohu bodu a tetrahedronu rozdělit na tři případy:

1. bod je uvnitř tetrahedronu,
2. bod leží na stěně tetrahedronu,
3. bod leží na hraně tetrahedronu.

Existují tedy tři různé způsoby dělení tetrahedronu. Popišme si je proto nyní podrobně.

Nachází-li se bod uvnitř tetrahedronu, pak každá z jeho čtyř stěn poslouží jako základna jednoho ze čtyř nových tetrahedronů. Každá hrana poslouží jako základna pro šest nových trojúhelníků, které tvoří boční stěny nových tetrahedronů. Zmíněný případ blíže ilustruje obrázek 2.3 vlevo. Tento způsob dělení má tu výhodu, že nepřesáhne hranice tetrahedronu, a nezpůsobí tak další dělení. Shrnutí: 1 starý tetrahedron zrušen, 4 nové tetrahedrony vytvořeny, 4 staré trojúhelníky použity, 6 nových trojúhelníků vytvořeno.



Obrázek 2.3: Způsoby dělení vycházející z různé polohy bodu a tetrahedronu: bod leží uvnitř tetrahedronu, na stěně tetrahedronu, na hraně tetrahedronu

Leží-li bod na stěně tetrahedronu, jedná se o speciální konfiguraci předchozího případu – jeden ze čtyř tetrahedronů degeneruje na čtyři body v rovině, které samozřejmě žádný tetrahedron netvoří. Základní tetrahedron je tak rozdělen pouze na tři nové, čímž vzniknou tři nové vnitřní stěny. Jak je zřejmé z obrázku, dochází ještě k rozdělení jedné základní stěny na nové tři. Tato skutečnost s sebou přináší menší komplikaci, a sice nutnost rozdělit stejným způsobem tetrahedron sousedící přes právě rozdělenou stěnu (pokud ovšem existuje – viz obrázek 2.3 uprostřed). Shrnutí (případ se dvěma tetrahedrony): 2 staré tetrahedrony zrušeny, 6 nových tetrahedronů vytvořeno, 6 starých trojúhelníků použito, 1 starý trojúhelník zrušen, 9 nových trojúhelníků vytvořeno.

Leží-li bod na hraně tetrahedronu, je to opět speciální případ předchozích konfigurací. Z původních čtyř nových tetrahedronů zůstanou pouze dva, jak naznačuje obrázek 2.3 vpravo. Protože bod dělí jednu z hran, musí dělit i stěny (trojúhelníky), které tuto hranu sdílejí. To s sebou přináší větší komplikaci. Zatímco v předchozím případě byly děleny pouze dva tetrahedrony – neboť stěna může být sdílena nejvýše dvěma – v tomto případě nastává dělení neznámého počtu tetrahedronů, neboť hranu jich může sdílet libovolný počet. Dělení každého z tetrahedronů s sebou samozřejmě přináší dělení sdílených stěn a vytváření dalších stěn uvnitř tetrahedronů.

Shrnutí (pro K tetrahedronů sdílejících hranu): K starých tetrahedronů zrušeno, $2K$ nových tetrahedronů vytvořeno, K starých trojúhelníků zrušeno, $3K$ nových trojúhelníků vytvořeno.

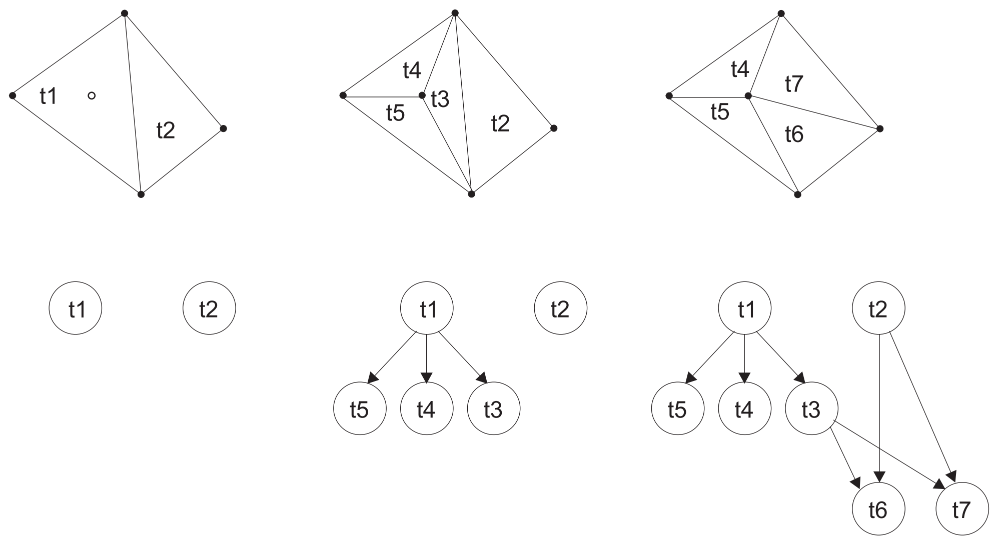
2.1.2 Vyhledání tetrahedronu

Naivním přístupem, tedy přístupem bez použití speciálních datových struktur, budeme vždy prohledávat všechny již vytvořené tetrahedrony. Protože tetrahedronů může být až $\mathcal{O}(N^2)$, jak si ukážeme dále v kapitole 2.2, byla by časová složitost vyhledání rovna $\mathcal{O}(N^2)$.

Lepšího výsledku dosáhneme použitím struktury *DAG*, orientovaného acyklického grafu (*directed acyclic graph*), který je ve své podstatě podobný hierarchické stromové struktuře. Tetrahedrony zorganizujeme velmi jednoduše tak, jak jsou vytvářeny. Nerušíme tedy tetrahedrony, které jsou děleny. Naopak. Uchováváme v jejich datových strukturách odkazy na přímé potomky, kteří vzniknou rozdělením nebo lokální transformací (o lokálních transformacích je pojednáno dále v části 2.3) – tedy odkazy na dva, tři nebo čtyři nové tetrahedrony. Pokud hledáme tetrahedron, v němž se daný bod nachází, zjistíme nejprve, ve kterém z potomků se nalézá (vždy právě v jednom; v případě sdílení stěny či hrany vybereme jednoho z nich – zde je nutné mít informace o sousednosti tetrahedronů, neboť dělení se může rozšířit z jednoho potomka do druhého, o tom však blíže až v implementační části) a dále obdobně prohledáváme pouze hierarchickou strukturu tohoto potomka.

Pokud jsou body rozmístěny rovnoměrně náhodně, je stromová struktura přibližně vyvážená a dosáhneme logaritmické časové složitosti vyhledávání pro jeden bod. V případě nevyváženosti, kdy strom degeneruje až na strukturu podobnou spojovému seznamu, čímž prohledáváme opět většinu vytvořených tetrahedronů, dostaneme časovou složitost vyhledávání $\mathcal{O}(N^2)$.

Jako ilustrace použití DAG struktury poslouží obrázek 2.4. Pro lepší přehlednost a bez újmy na obecnosti je použita analogie ve 2D – stromová organizace při inkrementální triangulaci. Nejprve je vložen bod, poté je patřičný trojúhleník rozdělen a nakonec je provedena lokální optimalizace.



Obrázek 2.4: Použití DAG struktury pro urychlení vyhledávání ve 2D

2.1.3 Obecná množina bodů

Nyní zbývá odstranit předpoklad z úvodu, a sice že všechny body jsou uzavřeny v jednom tetrahedronu. Existují v zásadě dva přístupy:

1. pokud takový tetrahedron neexistuje, vytvoříme ho,
2. vytvoříme více těchto základních tetrahedronů za pomoci konvexní obálky.

V prvním případu musíme vytvořit takový tetrahedron, který uzavře všechny zadané body. Toho dosáhneme přidáním nových čtyř bodů. Jejich souřadnice volíme tak, aby byly dostatečně vzdáleny od bodů původní množiny. Nesmí být ale příliš vzdáleny, jinak by při implementaci mohlo docházet k problémům s numerickou přesností. Obvykle jsou souřadnice těchto bodů voleny jako $[5M, 0, 0]$, $[0, 5M, 0]$, $[0, 0, 5M]$, $[-5M, -5M, -5M]$, kde M je hodnota získaná jako absolutní hodnota extrému ve směrech x, $-x$, y, $-y$, z, $-z$.

Poté již postupujeme standardním způsobem dělení tetrahedronů postupným vkládáním bodů, dokud nejsou všechny body vybrány. Protože je použita klasická stromová struktura, probíhá vyhledání v logaritmickém čase a celková časová složitost vyhledávání pro N bodů je $\mathcal{O}(N \log N)$. Na závěr je však ještě nutné odstranit přidané čtyři body. S tím je spojeno odstranění tetrahedronů, které tyto body obsahují, tak, aby síť zůstala „nepoškozená“.

Druhý přístup se vyhýbá odstraňování ohraničujícího tetrahedronu tím, že za přirozené ohraničení bodů považuje jejich konvexní obálku. Z ní je vytvořena první vrstva ohraničujících tetrahedronů. Tyto tetrahedrony mají za základny stěny (trojúhelníky) konvexní obálky, čtvrtý vrchol je společný všem těmto tetrahedronům a je to libovolně vybraný bod, který nepatří do konvexní obálky. Pokud takový bod neexistuje, provede se tetrahedronizace konvexního mnohostěnu. Hledání tetrahedronu, v němž se nalézá daný bod, je prováděno nejprve postupně ve všech tetrahedronech první vrstvy, poté již hierarchicky v jednom z nich. Takovéto prohledávání – zřejmě závisející na počtu bodů a stěn konvexní obálky K – má časovou složitost $\mathcal{O}((N - K)(K + \log N))$. K celkové složitosti algoritmu je pak nutno přičíst ještě čas potřebný na vytvoření konvexní obálky, který je optimálně $\mathcal{O}(N \log N)$.

2.2 Počet vytvořených tetrahedronů

Při triangulaci množiny bodů ve dvourozměrném prostoru je počet vytvořených trojúhelníků závislý pouze na celkovém počtu bodů a na počtu bodů konvexní obálky dané množiny, přičemž tato závislost je lineární. Přesněji řečeno platí vztah

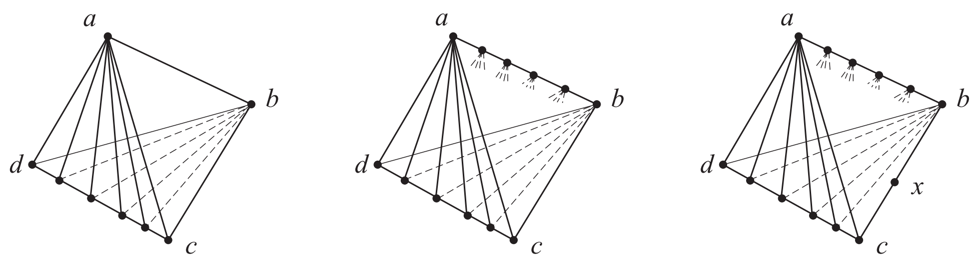
$$t = 2n - k - 2,$$

kde t značí počet trojúhelníků, n celkový počet bodů, k počet bodů konvexní obálky. Ve třírozměrném prostoru bohužel nejen že tato závislost neplatí, ale počet tetrahedronů se může měnit lineárně až kvadraticky v závislosti na daném počtu bodů, viz [AEG87]. Čím je to způsobeno?

Použijme opět zjednodušující předpoklad, že množina bodů je jednoduchá (čtyři body z dané množiny tvoří tetrahedron, ve kterém jsou zbylé body uzavřeny). Budeme vycházet z předchozího popisu inkrementálního algoritmu. Pokud by se nově vkládané body nacházely pouze uvnitř tetrahedronů – první uvedený způsob dělení (tzn. žádné čtyři body neleží v rovině a žádné tři body na přímce), pak by se s každým vloženým bodem počet tetrahedronů zvýšil o tři, pro N bodů by to tedy bylo $3 * (N - 4) + 1$ neboli $3 * N - 11$ (pro $N \geq 4$) tetrahedronů. Jde o lineární závislost. Připusťme, že čtyři body mohou ležet v rovině, tedy na stěnách tetrahedronů (druhý způsob dělení) – přidáním každého dalšího bodu je počet tetrahedronů zvýšen sice o čtyři, stále však zůstáváme u lineární závislosti $4 * (N - 4)$ (pro $N \geq 5$).

Je zřejmé, že příčinu kvadratické závislosti počtu tetrahedronů na počtu vstupních bodů musíme hledat v kolineárních bodech.

Představme si množinu o $N+2$ bodech, která je na obrázku 2.5 vlevo. Na hraně cd leží N bodů, které postupným vkládáním vytvoří $N-1$ tetrahedronů. Libovolný bod ležící na hraně ab rozdělí každý z existujících tetrahedronů na dva nové, což spolu se zrušením původního tetrahedronu zvýší jejich celkový počet o $N-1$. Přidáme-li takto na hranu ab dalších $N-3$ bodů (celkem tato hrana obsahuje N bodů – viz obr. 2.5 uprostřed), získáme pro množinu $2N$ bodů, $(N-1)^2$ tetrahedronů.



Obrázek 2.5: Ilustrace vzniku kvadratické závislost počtu tetrahedronů

Představme si dále, že do výše uvedené množiny $2N$ bodů, přidáme další bod x ležící na hraně bc – viz obr 2.5 vpravo. Pokud budeme základní tetrahedron dělit nejprve podle bodu x a potom podle ostatních bodů, získáme dva tetrahedrony $axcd$ a $abxd$, které budou dále rozděleny, každý na $N-1$ tetrahedronů. Celkový počet tetrahedronů bude $2(N-1)$. Pokud však pro dělení použijeme nejprve body na hranách, získáme $(N-1)^2$ tetrahedronů. Přidáním bodu x jako posledního vytvoříme navíc jeden tetrahedron.

Shrňme tedy dosažené poznatky: Množina N bodů ($N \geq 5$) v E^3, kde žádné tři body neleží na přímce, může být triangularizována za použití nejvýše $4(N-4)$ tetrahedronů. Existují množiny $2N+1$ ($N \geq 2$) bodů v R^3, které mohou být triangularizovány nejméně $2(N-1)$ a nejvýše $(N-1)^2+1$ tetrahedrony.

Je zjevné, že o tom, zda bude počet vytvořených tetrahedronů záviset na počtu bodů lineárně nebo kvadraticky, rozhoduje nejen počet kolineárních bodů, ale též pořadí, v němž budou body do triangularizace vkládány.

2.3 3D Delaunayova triangulace s použitím lokálních transformací

Delaunayova triangulace ve 2D je jedním z oblíbených postupů, jímž dosáhnout „dobrých“ tvarů trojúhelníků. Aby triangulace byla Delaunayovská, musí splňovat podmínku prázdné kružnice opsané – žádný z vytvořených trojúhelníků nesmí ve své kružnici opsané obsahovat bod ze vstupní množiny. Tato, na první pohled nic neříkající vlastnost, má tentýž význam jako podmínka maximalizace minimálního úhlu trojúhelníků v síti. Prakticky se jedná o vytváření sítě, jejíž elementy jsou co možná nejrovnostrannější.

Abychom mohli zkonstruovat 3D Delaunayovu triangulaci dříve uvedeným postupem inkrementálního algoritu, je třeba použít další myšlenku – *lokální transformace*, jak jsou uvedeny v [Joe91]. Lokální transformace slouží k lokální optimalizaci sítě. Delaunayova tetrahedronizace je síť lokálně optimalizovaná na kritérium prázdné koule opsané, shodou okolností se jedná i o kritérium globální. Pokud chceme dosáhnout lokálně optimální sítě, musíme použít princip lokálních transformací po každém kroku inkrementálního vkládání – tedy po vložení bodu a rozdělení tetrahedronu je nutno síť zoptimalizovat.

Použití lokálních transformací je naznačeno v části 2.3.2, jejich využítí s dříve popsaným inkrementálním algoritmem pak v části 3.1.

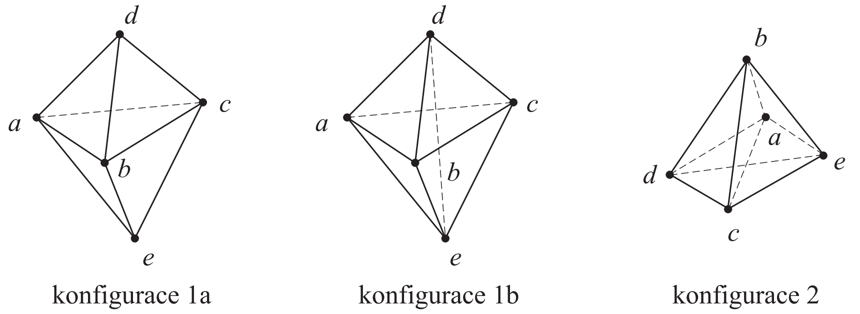
Z pohledu implementace jsou lokální transformace výhodnější než alternativní metody tvorby Delaunayovy tetrahedronizace (viz [Wat97] nebo [AEG87]), neboť vykazují vyšší robustnost a stabilitu – díky zaokrouhlovacím chybám v numerických výpočtech sice může dojít k chybnému vyhodnocení některých konfigurací, tím i k lokální chybě a lokálně neoptimální síti. Vzhledem však k tomu, že transformace pouze zaměňuje tetrahedrony uvnitř pevného objemu, nedochází k vytváření neplatných triangulací s překrývajícími se tetrahedrony či naopak s děrami – viz [Joe91].

2.3.1 Lokální transformace

Lokální transformace je postup založený na možných konfiguracích pěti navzájem různých nekomplanárních (neležících v jedné rovině) bodů. Popišme nejprve všechny možné konfigurace.

Konfigurace 1 Žádné čtyři body nejsou komplanární a konvexní obálka obsahuje všech pět bodů. V tomto případě je možno vnitřek vzniklého šestistěnu triangulovat dvěma způsoby (hrana de protíná vnitřek trojúhelníka abc) – viz obr. 2.6 vlevo a uprostřed.

1. vzniknou dva tetrahedrony $abcd$ a $abce$ nebo
2. vzniknou tři tetrahedrony $abde$, $acde$ a $bcde$.

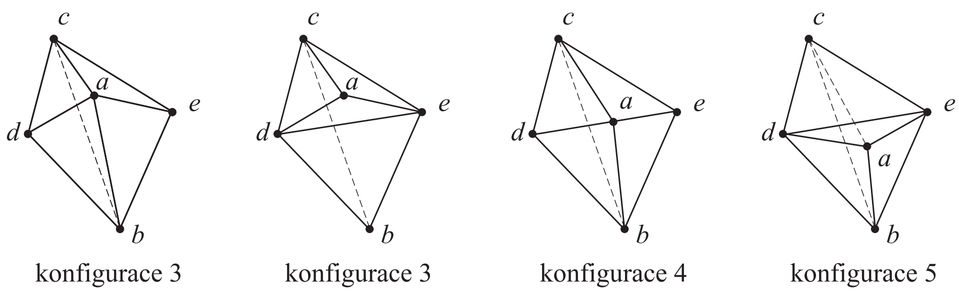


Obrázek 2.6: Možné konfigurace pěti bodů, žádné čtyři body nejsou komplanární

Konfigurace 2 Žádné čtyři body nejsou komplanární a konvexní obálka obsahuje čtyři z pěti bodů. CH tedy tvoří tetrahedron ve kterém leží pátý bod. V tomto případě vzniknou čtyři nové tetrahedrony $abcd$, $abce$, $abde$ a $acde$ – viz obr. 2.6 vpravo.

Konfigurace 3 Čtyři body jsou komplanární a tvoří striktně konvexní čtyřúhelník. Vnitřek jehlanu je možno triangularizovat dvěma způsoby (viz obr. 2.7 první dva zleva):

1. vzniknou dva tetrahedrony $abcd$ a $abce$ nebo
2. vzniknou dva tetrahedrony $acde$ a $bcde$.



Obrázek 2.7: Možné konfigurace pěti bodů, čtyři body jsou komplanární

Konfigurace 4 Čtyři body jsou komplanární, ale čtyřúhelník degeneroval na trojúhelník. Vzniknou tak dva tetrahedrony $abcd$ a $abce$ – viz obr. 2.7 třetí zleva.

Konfigurace 5 Čtyři body jsou komplanární a tvoří trojúhelník s jedním bodem uzavřeným uvnitř. Žádné tři body nejsou kolineární. Vzniknou tři tetrahedrony $abcd$, $abce$ a $acde$ – viz obr. 2.7 vpravo.

Mějme triangulaci daných pěti bodů. *Lokální transformací* pak nazveme postup, kdy – pokud jsou body v konfiguraci 1 nebo 3 – nahradíme tuto tetrahedronizaci druhou možností. Tento proces lze také popsat jako prohazování stěn (obdobou ve 2D je prohazování úhlopříček), kdy vnitřní stěnu abc v konfiguraci 1 zaměníme za stěny ade, bde, cde nebo naopak, v konfiguraci 3 zaměníme vnitřní stěnu abc za stěnu cde nebo naopak.

Kdy je možno transformovat? Mějme dva tetrahedrony $abcd$ a $abce$, které sdílejí stěnu abc a zároveň body a, b, c, d, e jsou v konfiguraci 1 nebo 3.

V konfiguraci 1 je stěna abc transformovatelná, pokud jsou oba tetrahedrony buď v konfiguraci 1a (pomyslná hrana de protíná vnitřek trojúhelníka abc) nebo v konfiguraci 1b, kde $abcd \bigcup abce$ není konvexní a v dané triangulaci existuje třetí tetrahedron potřebný k zaplnění konvexní obálky (roli stěny abc přebírá buď stěna dea nebo deb nebo dec). Pokud tento tetrahedron neexistuje, potom žádná ze stěn dea, deb, dec není transformovatelná, tj. nemůžeme použít lokální transformaci. Transformaci tedy nelze provést v případě, kdy společný soused prvních dvou tetrahedronů není shodný, neboť tehdy by konfigurace obsahovala více než tři tetrahedrony a tedy nutně více než pět bodů.

V konfiguraci 3 je stěna abc (nebo stěna dec) transformovatelná tehdy, jestliže existuje takový bod f na opačné straně striktně konvexního polygonu $adbe$ než bod c tak, že existují tetrahedrony $abdf$ a $abef$. Pokud sousedící tetrahedrony přes stěny abd a abe jsou $abdf$ a $abeg$, kde $f \neq g$, neni stěna abc transformovatelná.

2.3.2 Konstrukce sítě

Protože se jedná o konstrukci 3D Delaunayovy triangulace, je na vytvářené tetrahedrony kladeno kritérium prázdné opsané koule. Tetrahedron je lokálně optimální, jestliže jsou všechny jeho stěny lokálně optimální – obecná stěna abc je lokálně optimální, pokud koule opsaná tetrahedronu $abcd$ neobsahuje uvnitř bod e sousedního tetrahedronu $abce$.

V konfiguraci 1 je optimální buď stěna abc nebo tři stěny abe, bde, cde (může nastat pouze jeden případ, pokud všech pět bodů neleží na kouli). Stěny abe, bde, cde jsou buď všechny lokálně optimální nebo žádná z nich není lokálně optimální. V kofiguraci 2 je všech šest vnitřních stěn abc, abd, abe, acd, ace, ade lokálně optimálních. V konfiguraci 3 je lokálně optimální buď stěna abc nebo cde. V kofiguraci

4 je vnitřní stěna *abc* lokálně optimální, stejně tak jako v konfiguraci 5 tři vnitřní stěny *abc*, *acd*, *ace*.

Nyní k vlastnímu postupu. [Joe91] používá další z možných variant inkrementálního algoritmu – nebuduje síť „zvnějšku“ pomocí hraničního tetrahedronu nebo konvexní obálky za užití procesu dělení, nýbrž „zevnitř“ postupným obalováním již existující sítě. Nejprve je zformován první tetrahedron tak, že uvnitř není obsažen žádný ze zbývajících bodů. To je ostatně obecná podmínka tohoto algoritmu – i-tý bod přidávaný k existující triangulaci $i-1$ bodů musí ležet vně konvexní obálky těchto $i-1$ bodů. Síť tetrahedronů je vytvářena postupným přidáváním bodů, formováním příslušných tetrahedronů a následné optimalizace vnitřních stěn, dokud nedosáhneme Delaunayovy triangulace i bodů. Pak se celý postup od přidání bodu opakuje.

Při přidání dalšího bodu zjistíme všechny okrajové stěny stávající triangulace, které jsou z jeho pozice viditelné (viditelné jsou tehdy, když vnitřní bod triangulace a přidáváný bod leží na opačné straně roviny) a tyto stěny použijeme jako základny pro tetrahedrony s vrcholy v přidávaném bodě. Pro optimalizaci stěn je použita pomocná datová struktura, nejčastěji zásobník nebo fronta. Do ní jsou ukládány všechny stěny, které je nutno překontrolovat na podmínku lokální optimality. V tomto kroku jsou do zásobníku (fronty) vkládány základny nových tetrahedronů, tj. stěny které prošly testem na viditelnost. Po zformování všech nových tetrahedronů z viditelných stěn a přidaného bodu, jsou uložené stěny ze zásobníku (fronty) postupně vybírány. Pokud nejsou lokálně optimální a jsou transformovatelné, provede se lokální optimalizace v závislosti na konfiguraci bodů. Tím, že jsme lokálně zoptimalizovali vnitřní stěny útvaru ze dvou, tří nebo čtyř tetrahedronů, mohli jsme porušit kritérium opsané koule pro vnější stěny tohoto útvaru. Musíme je tedy znovu zkontrolovat – pokud již nejsou v zásobníku (frontě) obsaženy, uložíme je tam. Tato lokální optimalizace probíhá po přidání každého bodu, vždy až do vyprázdnění zásobníku (fronty), neboť teprve tehdy dosáhneme lokálně optimální – Delaunayovy – triangulace i bodů.

Časová složitost algoritmu, jak ji uvádí autor, je pro nejhorší možný případ (počet tetrahedronů $\mathcal{O}(N^2)$ $\mathcal{O}(N^2)$, očekávaná složitost pak $\mathcal{O}(N^{4/3})$ nebo $\mathcal{O}(N(\log N)^2)$.

V literatuře je popsáno několik alternativních metod pro konstrukci 3D Delaunayovy triangulace. V další části bude zmíněn Watsonoův algoritmus, což je alternativní postup k lokálním transformacím. Jako další postup bude představena konstrukce 3D Delaunayovy triangulace v lineárním čase.

2.4 Watsonův algoritmus

Watsonův algoritmus pro konstrukci Delaunayovy triangulace je alternativním postupem k již uvedenému inkrementálnímu algoritmu. Procesy dělení tetrahedronů a lokálních transformací jsou zde nahrazeny retriangulací určité podmnožiny již vytvořené sítě. Watsonův algoritmus je uveden např. v [GD97].

Představme si dříve uvedenou situaci, kdy postupně vkládáme body do již existující sítě. Po přidání jednoho bodu je nutno určit všechny tetrahedrony, které ve své opsané kouli obsahují daný bod. Jedná se právě o ty elementy, které porušují Delaunayovu podmínku – musí být tedy ze stávající sítě odstraněny. Tím je v ní vytvořena „díra“, kterou je nutno následně znovu triangularizovat. Za tímto účelem je vytvořen seznam všech stěn, které toto prázdné místo obklopují. Retriangulace je pak provedena spojením vloženého bodu s každou stěnou seznamu. Tím je vytvořena nová množina tetrahedronů, která splňuje Delaunayovu podmínku.

Nevýhodu tohoto algoritmu z časového hlediska je nutnost vyhledání všech tetrahedronů obsahujících vkládaný bod ve své opsané kouli. Z pohledu implementace, jak je popisováno v [Joe91], se jako hlavní slabina zmíněného algoritmu jeví singulární případy, kdy více než čtyři body leží na jedné kouli. V těchto případech může díky zaokrouhlovacím chybám při práci s čísly v pohyblivé řádové čárce dojít ke špatné retriangulaci vzniklého prostoru. Výsledkem je invalidní triangulace obsahující překrývající se tetrahedrony nebo naopak netriangularizované díry.

2.5 3D Delaunayova triangulace v lineárním čase

Hlavní předností této metody, jak je uvedená v [FP95], je očekávané lineární časové složitosti. Té je dosaženo díky minimalizaci časů potřebných pro vyhledávání tzv. Delaunayových vrcholů (viz dále).

2.5.1 Předzpracování

Základem pro urychlení vyhledávání je vytvoření trojrozměrné sítě buněk (kvádrů) nad danou množinou bodů tak, aby počet buněk přibližně odpovídal počtu bodů (rozměry buňek jsou odvozeny z celkového počtu bodů a minimálních a maximálních souřadnic bodů ve směrech os x, y, z). Každý bod je pak – na základě své pozice – přiřazen právě do jedné buňky. Buňka může obsahovat více bodů, v průměru však připadá na každou buňku jeden bod. Vyhledávání Delaunayových vrcholů je prováděno prohledáváním buněk vybíraných s ohledem na tzv. ohraničující kvádry, které určují maximální velikost prohledávané oblasti (pokud chceme např. prohledat buňky uvnitř koule, označíme za ohraničující kvádr nejmenší kvádr obsahující tuto kouli).

2.5.2 První tetrahedron

Protože od prvního tetrahedronu je dále konstruována celá síť, je nutné, aby bezpodmínečně splňoval podmínku Delaunayova kritéria. Jinak by konečná síť nebyla korektní. Proto je první tetrahedron vytvářen postupně. První bod může být libovolný, většinou se vybírá některý ze středové buňky nebo buňky v jejím okolí, aby síť rostla rovnoměrně ve všech směrech. Druhý bod – nejbližší bod k prvnímu (neboli koule – s průměrem rovným vzdálenosti prvního a druhého bodu – opsaná těmto dvěma bodům neobsahuje žádný jiný bod) nalezneme buď ve stejné buňce, nebo prohledáním buněk sousedních (od bližších ke vzdálenějším). Tím jsme dostali první hranu a můžeme zkonstruovat první trojúhelník. I pro ten musí platit podmínka opsané koule. Vybereme opět některý z blízkých bodů a vytvoříme kouli opsanou vzniklému trojúhelníku. Nyní musíme prohledat v ní uzavřené buňky, neboť mohou obsahovat „výhodnější“ třetí vrchol pro první trojúhelník. Změnou bodu je změněna i opsaná koule a přidány další buňky do seznamu prohledávaných. Nalezení čtvrtého bodu a konstrukce tetrahedronu je již obecným krokem, který je popsán v následující části.

2.5.3 Konstrukce sítě

Síť tetrahedronů je vytvářena postupným „obalováním“ již existující sítě, kdy k okrajovým stěnám (trojúhelníkům) jsou vyhledávány čtvrté tzv. Delaunayovy body tak, aby vytvořený tetrahedron splňoval kritérium opsané koule a byl tedy Delaunayův. Rychlé nalezení těchto bodů má výrazný vliv na výkonnost celého algoritmu. Existují dva takové způsoby prohledávání buněk:

1. *postupné prohledávání* je pomalejší, ale snadněji implementovatelné,

2. *tunelové prohledávání* je rychlejší, ale implementace je obtížnější.

Oba způsoby však využívají faktu, že čtvrtý bod k danému trojúhelníku může ležet pouze v jednom z poloprostorů určených rovinou trojúhelníku (nemůže ležet uvnitř již vytvořené sítě). K určení vnitřního a vnějšího směru musí být vrcholy každého trojúhelníku jednotně uspořádány (buď po směru nebo proti směru hodinových ručiček) tak, aby normálové vektory všech trojúhelníků ukazovaly vně nebo dovnitř vytvořené sítě.

Postupné prohledávání začíná v buňce, v níž leží těžiště krajního trojúhelníku. Oblast se postupně rozšiřuje všemi směry, přičemž se ještě upravuje na základě již nalezených bodů a ohraničujících kvádrů opsaných koulí. *Tunelová metoda* nejprve určí směr, který je dominantní pro normálu okrajového trojúhelníka ($\pm x$, $\pm y$, $\pm z$), a v tomto směru prohledává buňky v tunelech – od buňky již protíná trojúhelník směrem ven z již vytvořené sítě. Tento tunel je postupně rozšiřován (obalován dalšími vrstvami) a i zde je prohledávaná oblast upravována na základě již nalezených bodů.

Nalezením Delaunayova tetrahedronu však ještě není vše hotovo. V datových strukturách jsou uchovávány okrajové stěny, které budou dále rozšiřovány. Vytvořením nového tetrahedronu vyjmeme jeho základnu ze seznamu hraničních stěn a vložíme do něj tři nové (boční) stěny. Pokud však vytvoříme tetrahedron, který sousedí s jiným, již existujícím, tetrahedronem např. přes jednu boční stěnu, uzavřeme ji tím uvnitř sítě. Tuto stěnu již nesmíme v dalším postupu použít – je nutno ji vyjmout z datových struktur. Takové speciální případy mohou nastat tři a všechny je nutno ošetřit:

- dotyk jednou stěnou – vypustíme základnu a sdílenou stěnu, přidáme dvě nové stěny,
- dotyk dvěma stěnami – vypustíme základu a obě sdílené stěny, přidáme jednu novou stěnu,
- dotyk stěnou a hranou – přeskočíme vytvoření tohoto tetrahedronu tím, že vybereme další okrajovou stěnu ze seznamu (tetrahedron není možno vytvořit, protože by v síti vznikla uzavřená díra).

Jak již bylo uvedeno, je očekávaná časová složitost tohoto algoritmu lineární, v nejhorším případě (pro $\mathcal{O}(N^2)$ tetrahedronů) však může být až kvadratická. Z implementačního hlediska nelze algoritmus pokládat za příliš robustní, v singulárních případech nevykazuje dobrou stabilitu.

2.6 Hodnocení tvaru tetrahedronů

Různé 3D triangularizační algoritmy vytvářejí pro stejnou množinu bodů různé sítě, které se navzájem liší počtem tetrahedronů a samozřejmě též tvary tetrahedronů. Pokud nechceme vzájemné hodnocení sítí omezit na pouhé porovnání počtu vytvořených tetrahedronů, musíme se zabývat otázkou, jak mezi sebou porovnávat kvalitu jednotlivých tetrahedronů. Podívejme se na tetrahedron z geometrického hlediska a pokusme se vyhledat měřitelné atributy, které by mohly posloužit jako základ vzájemného porovnávání tetrahedronů, viz [LJ94].

Jistě můžeme vzít v úvahu takové vlastnosti jako délky hran, objem tetrahedronu a mnohé další, uvedené dále. Některé atributy jsou používány samostatně (např. velikosti úhlů), jiné je vhodné používat ve vzájemném vztahu (např. poloměr koule opsané vypovídá pouze cosi o velikosti tetrahedronu, zatímco v poměru s poloměrem koule vepsané získá další vypovídací hodnotu).

Po nalezení vhodných tvarových atributů a jejich vyčíslení je nutno určit, které z hodnot budeme považovat za lepší, a které za horší. V tomto směru nám většinou bude za „ideální“ tvar sloužit rovnostranný tetrahedron (všechny hrany mají stejnou délku – obdoba rovnostranného trojúheníku pro 2D). Vypočítané hodnoty nás pak informují o tom, jak „vzdálený“ je určitý tetrahedron od rovnostranného případu.

Protože v programové části jsou tvarová kritéria pojmenována v anglickém jazyce, bude i zde vedle českého termínu uveden vždy anglický ekvivalent názvu daného kritéria.

Objem (volume) nemá přesný význam ve smyslu uvedeném výše, tedy „velké hodnoty jsou lepší než malé“, neboť i malý rovnostranný tetrahedron může mít menší objem než velký asymetrický tetrahedron. Objem nás informuje o mohutnosti či velikosti tetrahedronu.

Objem čtyřstěnu, jehož jeden vrchol je v počátku kartézské soustavy souřadnic a ostatní tři vrcholy mají souřadnice $[x_i, y_i, z_i]$ pro $i = 1, 2, 3$, má objem rovný šestině absolutní hodnoty determinantu

$$V = \frac{1}{6}|D|, \qquad \text{kde} \quad D = \begin{vmatrix} x_1, & y_1, & z_1 \\ x_2, & y_2, & z_2 \\ x_3, & y_3, & z_3 \end{vmatrix}.$$

Tetrahedrony špatných tvarů mají většinou malý objem, neboť jsou často ploché či hubené a špičaté, zatímco rovnostranný tetrahedron je v prostoru rozložen rovnoměrě a zabírá tedy větší objem.

Poměr koule vepsané a opsané (radius ratio) Poloměr koule opsané vypovídá o velikosti tetrahedronu. V některých případech však – pokud je např. tetrahedron příliš plochý a je „namáčknut“ ke stěně koule opsané – ztrácí kritérium na vypovídací hodnotě. Poloměr koule vepsané naopak o plochosti vypovídá, není však invariantní vůči velikosti tetrahedronu. Poměrem těchto dvou poloměrů dostáváme kritérium, které informuje o míře rovnostrannosti tetrahedronu nezávisle na jeho velikosti.

Střed koule opsané tetrahedronu leží v průsečíku rovin půlících hrany, poloměr je roven vzdálenosti tohoto středu a libovolného vrcholu tetrahedronu. Střed koule vepsané se nachází v průsečíku rovin půlících vnitřní úhly mezi rovinami stěn, poloměr má hodnotu vzdálenosti tohoto středu od libovolné stěny. Výpočet hodnoty kritéria ρ je dán vztahem

$$\rho = \frac{3r}{R},$$

kde r je poloměr koule tetrahedronu vepsané a R je poloměr koule tetrahedronu opsané. Poloměr koule opsané lze vypočítat jako

$$r = \frac{3V}{\sum_{i=0}^{3} s_i},$$

kde V je objem a s_i povrchy stěn tetrahedronu. Poloměr koule vepsané získáme ze vztahu

$$R = \frac{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}{24V},$$

kde V je objem a a, b, c jsou součiny délek vždy dvou protilehlých hran tetrahedronu (těch, které nemají žádný společný bod).

Pro rovnostranný tetrahedron je hodnota tohoto kritéria rovna jedné (střed koule opsané je totožný se středem koule vepsané). Se změnou délek hran tetrahedronu dochází ke zmenšování koule vepsané a zvětšování koule opsané, hodnota poměru se zmenšuje. Kromě plochosti vypovídají nízké hodnoty tohoto kritéria i o výstřednosti tetrahedronu (střed koule opsané leží mimo tetrahedron).

Plocha minimálního Eulerova trojúhelníku (minimum solid angle) Plocha minimálního Eulerova trojúhelníka je logickým zobecněním minimálního vnitřního úhlu z 2D triangulací. Vyjadřuje, jak silně jsou stěny sevřeny kolem jednoho vrcholu.

Eulerův trojúhelník je pojem ze sférické trigonometrie. Pokud jeden vrchol tetrahedronu označíme za střed koule s jednotkovým poloměrem, protnou polopřímky hran vycházející z tohoto vrcholu jednotkovou kouli ve třech bodech. Spojením bodů oblouky vznikne na kouli Eulerův trojúhelník. Platí vztah

$$\sin(\theta_0/2) = \frac{12V}{\sqrt{\prod_{1\leq i<j\leq 3}(l_{0i}+l_{0j}+l_{lj})(l_{0i}+l_{0j}-l_{lj})}},$$

kde V je objem a l_{ij} délky hran tetrahedronu. Plocha minimálního Eulerova trojúhelníku θ_{min} je definována jako minimální plocha ze všech čtyř možných Eulerových trojúhelníků tetrahedronu. Platí $\sum i = 03\theta_i < 2\pi$.

Protože při výpočtu Eulerova trojúhelníku jsou používány cyklometrické funkce (arccos), které jsou výpočetně náročné, a protože přesnou hodnotu plochy trojúhelníku nepotřebujeme, neboť ji budeme užívat pouze pro porovnávání, lze použít následující trik. Nebudeme počítat hodnotu θ_{min}, ale $\sigma_{min} = min(\sigma_0, \sigma_1, \sigma_2, \sigma_3)$, kde $\sigma = sin(\theta_i/2)$. Že to lze, vyplývá z následující úvahy.

Předpokládejme $\theta_{min} = \theta_0 \leq \theta_1 \leq \theta_2 \leq \theta_3$. Protože $0 \leq \sum_{i=0}^{3}\theta_i \leq 2\pi$, pouze θ_3 může být větší než π, takže $\sigma_0 \leq \sigma_1 \leq \sigma_2$. Protože $\theta_2 \leq \theta_3 \leq 2\pi - \theta_2$, dostáváme $\sigma_2 \leq \sigma_3$. Takže

$$\sigma_{min} = \sin(\theta_{min}/2)$$

. Pro rovnostranný tetrahedron jsou $\theta_{min} = 0.5513$ a $\sigma_{min} = 0.2722$.

Poměr délek hran (edge ratio) Pokud za nejvhodnější tvar považujeme rovnostranný tetrahedron, je přirozené měřit odchylku od tohoto případu poměrem délek nejdelší a nejkratší hrany, který přímo informuje o míře nepravidelnosti hran tetrahedronu. Výpočet tohoto kritéria je zřejmý. Pro rovnostranný tetrahedron je výsledná hodnota rovna jedné, pro nepravidelné tetrahedrony je vždy větší než jedna. Čím větší hodnoty kritéria, tím větší rozdíly v délkách hran a tím „horší“ tvar tetrahedronu.

Poměr poloměru koule opsané a nejdelší hrany (aspect ratio) Jako jedno z poměrových kritérií není zavislé na velikosti tetrahedronu. Jedná se o další způsob měření plochosti tetrahedronu. Výpočet je zjevný z názvu kritéria. Pro ploché tetrahedrony s velkým poloměrem koule opsané – kdy je tetrahedron přitisknut k plášti koule – nabývá vysokých hodnot, naopak pro pravidelné tetrahedrony jsou hodnoty nízké.

2.7 Porovnání tetrahedronových sítí

V této části jsou popsány možné způsoby vzájemného porovnávání obecných sítí. Protože se nejedná o problém, jehož výsledkem jsou dvě čísla, která by bylo možno jednouše porovnat, používá se několik různých přístupů. Od porovnávání maximálních a minimálních hodnot kritérií, přes jejich porovnávání ve vektorech až k porovnávání histogramů.

2.7.1 Porovnání extrémů

Představme si, že máme vytvořeny dvě tetrahedronové sítě a chceme vybrat tu, která je pro nás nějakým způsobem výhodnější – lépe vyhovuje našemu hodnotícímu kritériu. Je-li tímto kritériem např. velikost plochy minimálního Eulerova trojúhelníku celé triangulace, spočteme jednoduše příslušnou hodnotu pro každý tetrahedron v síti a za lepší prohlásíme tu síť, pro niž je získaný výsledek větší nebo menší (záleží na tom, zda dané kritérum maximalizujeme či minimalizujeme, neboli zda považujeme za lepší větší či menší hodnoty).

2.7.2 Porovnání vektorů

Dříve uvedené řešení s porovnáním extrémů selže v případě, kdy obě triangulace mají hodnotu tohoto extrému stejnou – na jejím základě nemůžeme rozhodnout, která ze sítí je lepší. Proto je nutno pro každou ze sítí vytvořit vektor hodnot, jehož délka je rovna počtu elementů dané sítě. Prvky vektorů jsou seřazeny – vzestupně či sestupně v závislosti na tom, zda dané kritérium maximalizujeme či minimalizujeme.

Seřazené prvky ve vektorech pak postupně porovnáváme následujícím způsobem: je-li hodnota stejnolehlých prvků shodná, porovnáváme další dva prvky; je-li hodnota prvku v jednom vektoru menší (pro maximalizaci) nebo větší (pro minimalizaci) než hodnota prvku ve druhém vektoru, prohlásíme druhou triangulaci za lepší. Pro názornost uveďme příklad dvou triangulací s částí jejich vektorů hodnot, pro maximalizaci zadaného kritéria:

```
první triangulace: 1 4 8...
druhá triangulace: 1 3 5...
```

Za lepší prohlásíme první triangulaci, neboť při maximalizaci vítězí ta, která jako první nabude vyšší hodnoty. Pokud bychom použili obdobný příklad pro minimalizaci hodnoty kritéria,

```
první triangulace: 8 4 1...
druhá triangulace: 8 3 1...
```

konstatovali bychom, že lepší je druhá triangulace, neboť jako první dosáhne nižší hodnoty.

Protože vektory hodnot mohou být pro tutéž množinu bodů různě dlouhé (různý počet tetrahedronů), pak – pokud jsme již vybrali všechny prvky z jednoho vektoru – prohlásíme tuto triangulaci za lepší, neboť dosáhla stejných tvarových výsledků s menším počtem tetrahedronů v síti.

2.7.3 Histogramy

Vyneseme-li výsledky získané předchozím způsobem do grafu, získáme histogram vyjadřující rozložení vypočtených hodnot v celé síti. Oproti předchozímu způsobu, kde o lepší a horší síti rozhodují dvě rozdílné hodnoty, je tento způsob založen na vizuálním porovnání výsledných grafů. Tak je možno jednoduše srovnat tvary křivek stejného kritéria pro různé sítě, jejich změny, rozložení extrémů apod.

Uvedením tvarových kritérií a vzájemného porovnávání tetrahedronových sítí je obsah teoretické části vyčerpán. V další části následuje popis implementovaného algoritmu. Uvedeny budou očekávané předpoklady spolu s problémy, které se během implementace vyskytly, představeno bude rovněž nalezené řešení.

Kapitola 3

Implementace

Tato část je věnována metodě použité k vytváření tetrahedronových sítí. a popisu její implementace. Nejprve bude představen obecný přístup vycházející z nastudované teorie, který byl následně do jisté míry ovlivněn a směrován dosaženými praktickými výsledky, odvozena bude časová složitost algoritmu. Poté budou zmíněny datové struktury užité pro jednotlivé základní objekty a jejich vzájemné propojení. Dále budou podrobně popsány klíčové kroky celé implementace: dělení tetrahedronů během inkrementálního algoritmu, lokální transformace zajišťující lokální optimalitu vytvářené sítě a způsob zjišťování konfigurací tetrahedronů.

Po uvedení těchto zásadních programových akcí bude nastíněna celková struktura programu a dekompozice do jednotlivých souborů.

Výsledný program je součástí širšího projektu – MVE (modular visualization environment). Proto bude rovněž popsáno začlenění vytvořené aplikace do tohoto projektu. Přehledově budou zmíněny jak filosofie celého projektu a datové struktury, tak navazující moduly, např. generátor bodů, modul pro vizualizaci isoploch apod.

3.1 Vlastní metoda řešení

Hlavním úkolem této práce je vytvořit nad danou množinou bodů několik různých tetrahedronových sítí několika různými způsoby a na základě získaných výsledků mezi sebou porovnat jak vzniklé sítě, tak použité postupy.

3.1.1 Základní postup

K vyřešení zadaného problému by bylo možné použít několik různých postupů, z nichž každý bude vytvářet od počátku do konce požadovanou 3D triangulaci podléhající určitým požadavkům na vytvářené tetrahedrony. Jednalo by se o modifikaci některého z přímých algoritmů, kde bychom Delaunayovo kritérium nahradili libovolným jiným. Každý takový postup by však vyžadoval svoji analýzu s nalezením a ošetřením speciálních případů, jež by se mohly během výpočtu vyskytnout.

Dříve zmiňované lokální transformace jsou naopak jednotným postupem použitelným na libovolnou síť. Provedení lokální optimalizace závisí pouze na splnění podmínky zlepšení hodnoty vybraného kritéria. Jsou rovněž známy a popsány všechny možné konfigurace, na které lze lokální transformace použít. Tato jednotnost je dalším důvodem, proč byla lokálním transformacím dána přednost před několika samostatnými konstrukčními postupy.

Použitý algoritmus by měl být vhodným kompromisem mezi rychlostí a složitostí implementace – neměl by síť vytvářet naivně, ale použít pro urychlení další datové struktury. Během konstrukce by se nemělo vyskytnout příliš mnoho singulárních případů a rozhodování o řešení v těchto situacích by mělo být rychlé – těmto požadavkům inkrementální algoritmus rovněž vyhovuje. Ve spojení s lokálními transformacemi se jedná o elegantní způsob, kdy lze pouhou změnou optimalizačního kritéria nad stejnou množinou bodů stejným postupem získat jinou lokálně optimální síť.

Inkrementální algoritmus s lokálními optimalizacemi byl tedy zvolen za výchozí postup implementace. K tomuto výběru vedly, kromě již uvedených důvodů, také skutečnosti týkající se alternativních postupů – Watsonův algoritmus ve 3D je považován za horší postup, neboť produkuje větší množství tetrahedronů špatných tvarů (viz [GD97]). Algoritmus s očekávanou lineární časovou složitostí pak byl již implementován v rámci předmětu Vybrané algoritmické metody Martinem Kročákem. Získané zkušenosti nehovoří příliš ve prospěch tohoto postupu.

3.1.2 Proč post-optimalizace?

U Delaunayovy tetrahedronizace je známo, že její konstrukce inkrementálním algoritmem spolu s lokálními transformacemi za použití hraničního tetrahedronu konverguje k vytvoření tetrahedralizované konvexní obálky původní množiny bodů. To je velmi výhodná vlastnost pro odstranění hraničního tetrahedronu a s tím spojených nepotřebných elementů. Tuto vlastnost však nemají všechna kritéria. Přesto bylo cílem naší snahy vyzkoušet výše uvedený základní postup i na jiných kritériích. Případná nekonvergence vede k situaci, kdy není možno jednoduše odstranit pouze trojúhelníky a tetrahedrony incidující s přidanými čtyřmi vrcholy. Snaha o nalezení algoritmu, který by tento problém uspokojivě vyřešil, vedla následujícím směrem.

Je zřejmé, že cílová triangulace, v níž by bylo možno odstranit nepotřebné elementy pouhým jejich vyjmutím z datových struktur, musí existovat, neboť taková triangulace je např. Delaunayova. Je potřeba najít takový algoritmus, který by „upravil“ stávající síť do vyhovující podoby, tedy vytvořil uvnitř tetrahedronizace konvexní obálku původní množiny bodů. K řešení je opět možno použít lokální transformace, neboť ty jsou právě tím prostředkem, který umožňuje přejít od jedné tetrahedronizace k jiné.

Naším cílem je najít takovou posloupnost lokálních transformací, která ze zadané sítě vytvoří síť vhodnou pro odstranění hraničního tetrahedronu. Byly vyzkoušeny dva přístupy. Část společná oběma sestává z označení všech trojúhelníků, které musí být transformovány, neboť narušují podmínku konvexní obálky. To jsou ty, jejichž některý vrchol náleží do množiny čtyř přidaných vrcholů, některý z vrcholů leží v konvexní obálce původní množiny a některý z vrcholů je „obyčejný“. Chceme totiž získat síť, ve níž zůstanou jen trojúhleníky dvou typů:

- trojúhelníky incidující s některým z přidaných vrcholů musí mít všechny zbylé vrcholy obsaženy v konvexní obálce původní množiny, aby je bylo možno odstranit,
- ostatní trojúhelníky, jež neincidují s přidanými vrcholy.

Pokud máme identifikovány trojúhelníky určené k transformaci, můžeme začít s jejich odstraňováním pomocí lokálních transformací.

V prvním přístupu byly trojúhelníky transformovány následujícím způsobem:

```
opakuj, dokud není fronta prázdná
  vezmi první trojúhelník z fronty
  zjisti jeho konfiguraci
  pokud je transformovatelný
    proveď transformaci
  pokud není transformovatelný
    přesuň ho na konec fronty
```

Tento způsob se ukázal jako nevyhovující neboť na konci algoritmu zůstávaly ve frontě trojúhelníky, které nebyly transformovatelné – došlo k uváznutí algoritmu. Pokud však bylo u téže tetrahedronizace na počátku procesu zvoleno jiné pořadí vložení trojúhelníků do fronty, v některých případech bylo řešení nalezeno. Z výše uvedeného vyplývá jednoznačný závěr. Pořadí trojúhelníků rozhoduje o úspěšnosti celého algoritmu.

Závisí-li výsledek na pořadí transformací, jedná se o problém prohledávání grafu všech možných řešení, tedy o problém s exponenciální časovou složitostí. Algoritmy s touto složitostí nejsou v praxi použitelné. Většinou je hledána heuristika, tedy postup, který se snaží využít některých závislostí, a jímž se ve většině případů dostaneme k výsledku za použití menších časových nákladů. Taková heuristika byla hledána i v tomto případě.

Aby bylo zabráněno provedení těch změn v síti, které nakonec vytvoří netransformovatelné trojúhelníky, byl vlastní proces transformace uzavřen ještě další podmínkou. Je totiž nutno navíc otestovat, zda se při změně konfigurace jednoho trojúhelníku změní konfigurace trojúhelníků v konvexní obálce negativním způsobem. Tím je myšleno, že trojúhelník v konvexní obálce, který se před transformací nacházel v transformovatelné konfiguraci (tj. 1a, 1b nebo 3) se musí po transformaci nacházet opět v transformovatelné kofiguraci. Pokud tomu tak není, je původní transformace označena za předčasnou, není provedena a trojúhelník, jímž byla vyvolána, je zařazen na konec fronty.

Testování, zda byla konfigurace vyvolána předčasně či nikoliv, jinými slovy otestování všech stěn konvexní obálky na případné zhoršení je provedeno takto: nejprve jsou zjištěny konfigurace všech stěn konvexní obálky, tyto konfigurace jsou uloženy. Poté musí být otestován stav vzniklý po transformaci, což je možné provést pouze po transformaci – není možno testovat pouze vzájemné polohy bodů a trojúhelníků, protože v případě konfigurace 1b je nutno dohledávat v datových strukturách třetí chybějící tetrahedron. Proto je transformace provedena a trojúhelníky konvexní obálky jsou opět testovány na typ konfigurace. Pokud byl některý transformovatelný trojúhelník změněn na netransformovatelný, je tato změna zavržena jako předčasná, vše je vráceno do původního stavu a trojúhelník je zařazen na konec fronty. Jinak je nový stav po transformaci akceptován a celý postup od výběru dalšího trojúhelníku se opakuje.

Jak se však ukázalo, ani tento postup nevede ke získání kýžené triangulace. I zde může proces transformace uváznout, když ve frontě zůstanou pouze netransformovatelné trojúhelníky. Řešení problému tedy leží v oblasti prohledávání kompletního grafu všech možností. To znamená úlohu s exponenciální časovou složitostí, což je pro použití v reálném systému nemyslitelné. Proto jako další způsob, kterým je možno vytvářet různé lokálně optimální tetrahedronové sítě, byly zvoleny post-optimalizace.

3.1.3 Post-optimalizace

Proces *post-optimalizace* znamená vytvoření další lokálně optimální sítě (dalších lokálně optimálních sítí) z jedné výchozí. Můžeme ho také chápat jako pokus o „vylepšení“ vlastností původní sítě pomocí optimalizace vůči dalšímu speciálnímu kritériu. Tím se stále držíme původní myšlenky dosáhnout pomocí jednotného postupu (lokálních transformací) různé lokálně optimální sítě.

Realizace post-optimalizace probíhá následovně: předpokladem je již vytvořená libovolná tetrahedronizace nad zadanou množinou bodů, v našem případě se jedná o 3D Delaunayovu triangulaci, neboť ta nepostrádá vlastnost konvergence ke konvexní obálce původních bodů a s tím spojené jednoduché odstranění nežádoucích elementů. K dosažení tetrahedronizace lokálně optimální vůči některému z kritérií musíme znovu překontrolovat lokální optimalitu všech konfigurací nacházejících se ve zkonstruované síti. Toho dosáhneme tím, že do fronty uložíme všechny trojúhelníky. Nepoužijeme pouze ty v konvexní obálce, neboť ty nemohou být žádným způsobem transformovány. Poté postupujeme obdobně jako při optimalizaci během inkrementálního algoritmu: vybíráme postupně trojúhelníky z fronty a pokoušíme se je optimalizovat. Celý proces končí vyprázdněním fronty – dosáhli jsme lokálně optimální tetrahedronizace.

Ideově se post-optimalizace shoduje s optimalizací při konstrukci sítě, v datových strukturách ovšem probíhá jinak. Zatímco během inkrementálního algoritmu bylo zapotřebí uchovávat data v DAG struktuře (kvůli urychlení lokalizace bodu), po vytvoření Delaunayovy tetrahedronizace byla tato pomocná struktura odstraněna spolu hraničním tetrahedronem, přidanými čtyřmi body a z toho plynoucími nepotřebnými trojúhelníky a tetrahedrony. Post-optimalizace jsou tedy prováděny přímo nad „čistou“ sítí bez jakýchkoliv dalších pomocných struktur.

3.1.4 Post-optimalizační kritéria

Pokud jsme zvolili jako řešení zadaného problému lokální transformace použité při post-optimalizaci, musíme nyní zvolit i kritéria, vůči kterým bude proces vylepšování probíhat. Tato kritéria není nutno dlouho hledat. V teoretické části uvedená a tvary tetrahedronů hodnotící mohou posloužit stejně dobře i pro optimalizaci, neboť jsou-li schopna změřit kvalitu tetrahedronu, mohou být použita i pro její vylepšení.

Všechna uvedená kritéria lze aplikovat v zásadě dvěma způsoby. Oba si ukážeme na příkladu poměru poloměrů koule vepsané a opsané, u nějž jsou vyšší hodnoty známkou lepšího tvaru.

Při lokální optimalizaci, tedy transformaci jedné možné konfigurace na druhou, můžeme požadovat, aby za lepší byla prohlášena konfigurace s celkově nejvyšší hodnotou kritéria. To je tzv. *maximalizace* daného kritéria, již můžeme chápat jako pokus o nalezení největšího extrému v síti. Tento postup však často vede ke zhoršení extrémních hodnot na opačném pólu.

Druhý přístup prohlásí za lepší tu konfiguraci, která dosáhne větší minimální hodnoty. Je to tzv. *maximalizace minimální* hodnoty daného kritéria. Chápat ji lze jako pokus o vylepšení celkové kvality sítě tím, že se snažíme pozvednou dolní hranici hodnot. Výsledky dosažené tímto tímto způsobem by měly být dle očekávání kvalitnější.

Pro kritéria s opačnou logikou hodnot, kdy známkou lepšího tvaru jsou nižší hodnoty, se obdobným způsobem používá *minimalizace* a *minimalizace maximální* hodnoty daného kritéria.

Jako post-optimalizační kritéria typu maximalizace a maximalizace minimální hodnoty byla vybrána následující:

- poměr poloměrů koule opsané a vepsané (radius ratio),
- objem (volume).

Pouze pro maximalizaci hodnot byla – z logiky věci – zvolena kritéria

- plocha minimálního Eulerova trojúhelníku (minimum solid angle),
- počet tetrahedronů (toto kritérium nebylo zatím zmiňováno, jeho význam tkví ve výhradním provádění transformací 1a na 1b, tedy těch transformací, které zvyšují počet tetrahedronů v síti).

Pro minimalizaci a maximalizaci minimální hodnoty jsou použita kritéria

- poměr nejdelší a nejkratší hrany (edge ratio),
- poměr poloměru koule opsané a nejdelší hrany (aspect ratio).

Pouze pro minimalizaci hodnot bylo zvoleno kritérium

- počet tetrahedronů (minimalizace počtu tetrahedronů je prováděno výhradně transformacemi 1b na 1a, kdy se lokální transformací sníží počet tetrahedronů ze tří na dva).

3.1.5 Časová složitost algoritmu

Z možných alternativ inkrementálního algoritmu uváděných v teoretické části je dále použita varianta s vytvořením hraničního tetrahedronu. Tato metoda se zdála vhodnější z důvodu zachování principu jednotnosti celého algoritmu, kdy hlavní činnost bude seskupena do dvou základních částí – inkrementální algoritmus a lokální transformace. Nebude tedy nutno se zabývat konstrukcí konvexní obálky a „obohacovat“ tak celý postup o cizorodé prvky.

Při výpočtu časové složitosti algoritmu musíme mít stále na paměti, že počet tetrahedronů může být na počtu vstupních bodů závislý až kvadraticky.

Inkrementání algoritmus Použitím ohraničujícího tetrahedronu spolu s DAG strukturou získáme pro vyhledání tetrahedronu obsahujícího vkládaný bod časovou složitost $\mathcal{O}(\log N^2)$, tedy $\mathcal{O}(\log N)$. Rozdělení tetrahedronu proběhne v konstantním čase, pokud bod leží uvnitř nebo na stěně. Jestliže bod leží na hraně, může dojít k rozdělení až $\mathcal{O}(N)$ sousedních tetrahedronů (viz obr. 2.5). Toto číslo však nelze považovat za absolutní, neboť má úzký vztah k počtu následně prováděných lokálních transformací.

Lokální transformace Počet provedených lokálních transformací během jednoho kroku inkrementálního algoritmu závisí mimo jiné na celkovém počtu tetrahedronů v síti. Lze předpokládat, že v nejhorším případě bude nutno optimalizovat celou síť, tedy $\mathcal{O}(N^2)$ tetrahedronů znovu (proces lokální optimalizace je konečný, důkaz plyne z omezeného počtu všech potenciálně vytvořitelných triangulací nad danou množinou bodů). Složitost lokálních transformací je tedy v nejhorším případě $\mathcal{O}(N^2)$. Díky úzké souvislosti s dříve uváděným dělením tetrahedronu můžeme popsat případ (viz obr. 2.5), kdy, v případě dělení na $\mathcal{O}(N)$ tetrahedronů, bude počet optimalizovaných trojúhelníků pouze ve složitosti $\mathcal{O}(N)$. Existuje zde tedy závislost mezi počtem dělených a následně optimalizovaných tetrahedronů, díky níž může být vyčíslena celková složitost těchto dvou akcí jako $\mathcal{O}(N^2)$.

Výsledná složitost Celkovou časovou složitost lze odvodit z předchozích dvou uvedených. Inkrementální algoritmus je opakován pro všech N bodů. Pro každý tento bod provádíme vyhledání tetrahedronu v $\mathcal{O}(\log N)$, jeho dělení a lokální optimalizace celkově v $\mathcal{O}(N^2)$ – počet všech kroků provedených po vložení jednoho bodu tak může být maximálně $\mathcal{O}(N^2)$. Výsledná časová složitost algoritmu je potom $\mathcal{O}(N^3)$. Post-optimalizační proces na této složitosti již nic nezmění, neboť jeho časová složitost může být nejvýše $\mathcal{O}(N^2)$.

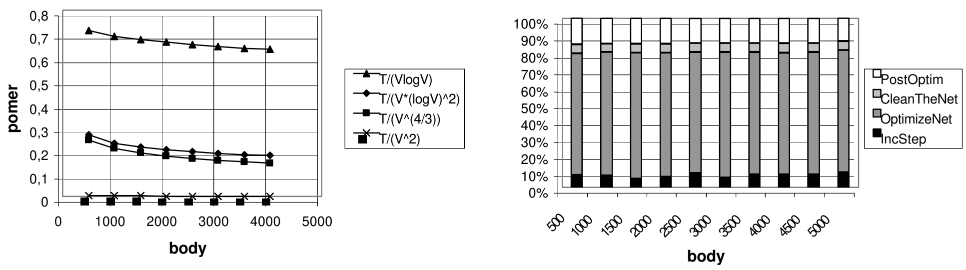
Očekávaná složitost Očekávanou časovou složitost předpokládáme lepší než právě odvozenou teoretickou složitost ze dvou důvodů:

1. počet tetrahedronů vzniklý při dělení a následně optimalizovaný je nepředpověditelný, neboť závisí na rozložení bodů,
2. nejhorší případy z hlediska časové složitosti nastávají, pokud mnoho bodů leží na jedné přímce; toto extrémní rozložení u reálných dat neočekáváme, předpokládáme spíše náhodně rozmístěné body.

Pokud budeme počítat s uvedenými předpoklady, můžeme počet dělených a následně optimalizovaných trojúhelníků považovat za lineární, což ostatně uvádí i [Joe91]. Počet trojúhelníků v post-optimalizačním procesu lze se stejnými předpoklady považovat rovněž za lineární. Časová složitost vyhledání tetrahedronu zůstává $\mathcal{O}(\log N)$. Očekávaná časová složitost celého algoritmu tak nabývá hodnoty $\mathcal{O}(N \log N)$.

Praktickými testy bylo ověřeno, že očekávaná složitost se skutečně blíží výše uváděné hranici. Na obr. 3.1 vlevo je graf sloužící k odhalení časové závislosti – na osu x je vynášen počet bodů, na osu y potom čas vydělený očekávanou závislostí na bodech. Pokud výsledná křivka grafu stoupá, znamená to, že složitost ve jmenovateli je příliš nízká. Klesá-li však výsledná křivka, byla zvolena horší složitost a můžeme se pokusit najít jinou.

Pro ilustraci časových poměrů v programu je uveden následující graf 3.1. Část `IncStep` označuje inkrementální algoritmus s vyhledáváním bodů a dělením tetrahedronů. `OptimizeNet` je čas strávený během optimalizací, tedy vylepšováním sítě lokálními transformacemi. Je patrné, že tato část programu zabírá dalece největší časový úsek. `CleanTheNet` provádí zrušení DAG struktury a odstranění nepotřebných elementů sítě. Poslední část `PostOptim` se věnuje vylepšování vytvořené Delaunayovy tetrahedronizace.



Obrázek 3.1: Očekávaná časová složitost a časové poměry v programu

3.2 Datové struktury

Jak vyplývá z teoretické části, je možno tetrahedronové sítě ukládat do různých datových struktur. Analogií z 2D triangulace, kde se používají struktury typu trojúhelník–vrcholy nebo trojúhelník–hrany–vrcholy (pojmenování vyjadřuje hierarchii jednotlivých objektů, vzájemné propojení záleží na jejich detailnějším pojetí), dospějeme ve 3D k reprezentaci typu tetrahedron–vrcholy, tetrahedron–trojúhelníky–vrcholy, tetrahedron–trojúhelníky–hrany–vrcholy.

Pro implementaci inkrementálního algoritmu se stromově organizovanými tetrahedrony se první typ nejeví zcela vhodný, neboť jednoduše neposkytuje možnost uložení normálových vektorů stěn tetrahedronu. Každý tetrahedron by musel mít uloženy čtyři normály svých stěn, což by vedlo jednak k jejich duplicitě – neboť jedna stěna je sdílena dvěma tetrahedrony – jednak by mohlo dojít k jejich odlišnostem vzhledem k výpočtům v pohyblivé řádové čárce. Typ tetrahedron–trojúhelník–hrany–vrcholy se díky účasti hran jeví zbytečně redundantní. Jako vhodná reprezentace dat pro uvažovanou implementaci se jeví struktura typu tetrahedron–trojúhelník–vrcholy.

3.2.1 Vrcholy

Tetrahedronová síť je vytvářena nad množinou vrcholů. Vrcholy jsou ve 3D prostoru zadány svými souřadnicemi, předpokládá se, že žádné dva vrcholy nejsou totožné. Počet vrcholů je předem znám – to je nutno jednak z hlediska výpočtu (potřebujeme vytvořit ohraničující tetrahedron) a jednak z principu funkčnosti MVE (každý modul má přesně určený vstup a výstup).

Vrcholy jsou ukládány do struktury `tVertex`, která obsahuje hodnoty všech tří souřadnic a proměnnou pro ukládání příznaků:

```
typedef struct {
  double x;  // coordinates
  double y;
  double z;
  long flags;
} tVertex;
```

Struktury všech vrcholů jsou uloženy v poli konstantní délky `VertexArray`, které je dynamicky alokováno na začátku programu. K jednotlivým vrcholům je pak přistupováno staticky přes jejich indexy uložení v poli.

3.2.2 Trojúhelníky

Informace o trojúhelníkách jsou uloženy ve struktuře `tTriangle`:

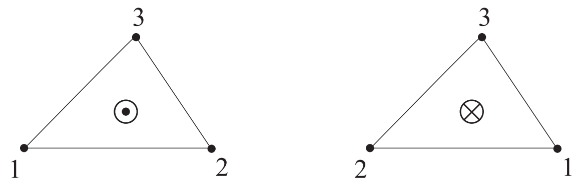
```
typedef struct sTriangle {
  long tn;  // element number
  struct sTriangle *next;
  struct sTriangle *prev;
  struct sTriangle *nextque;
  struct sTriangle *prevque;
  long flags;
  long v1;  // vertices of triangle
  long v2;
  long v3;
  tVector n;  // normal vector of plane
  double d;  // determines shift of plane
  struct sTetrahedron *th1;  // normal order of vertices -- normal vector points out
  struct sTetrahedron *th2;  // reversed order of vertices -- normal vector points in
} tTriangle;
```

Jak bylo uvedeno v teoretické části (viz 2.2), může být počet tetrahedronů (a tím i trojúhelníků) ve vytvářené síti závislý na počtu vrcholů lineárně až kvadraticky. Není tedy možné předem vypočítat ani jinak zjistit počet vytvořených trojúhelníků. Struktury trojúhelníků jsou proto organizovány jako obousměrně zřetězený spojový seznam pomocí ukazatelů `next` a `prev`.

Během optimalizačního procesu je třeba trojúhelníky organizovat do fronty, proto i tato struktura obsahuje další dva ukazatele `nextque` a `prevque` pro začlenění do obousměrně zřetězeného seznamu fronty. Trojúhelník dále obsahuje odkazy na svoje vrcholy `v1`, `v2` a `v3` do pole bodů. Klíčovou částí, která slouží pro rychlé testování vzájemné pozice trojúhelníku a bodu, jsou předpočítané a uložené parametry roviny trojúhelníku – normovaný normálový vektor `n` a parametr posunu roviny `d`.

Následují dva ukazatele na tetrahedrony `th1` a `th2`, neboť každý trojúhelník je sdílen dvěma tetrahedrony (pokud neleží v konvexní obálce, pak je ukazatel roven `NULL`). Pořadí tetrahedronů v těchto ukazatelích je důležité, neboť musí být v souladu se směrem normály trojúhelníku.

Trojúhelník je zadán třemi body, jejichž pořadí rozhoduje o směru normály. Pokud pohlížíme na trojúhleník shora a proti směru hodinových ručiček existuje pořadí vrcholů 123, ukazuje normála proti nám. Pokud je pořadí bodů obrácené, ukazuje normála směrem od nás. Oba případy znázorňuje obr. 3.2.



Obrázek 3.2: Pořadí bodů v trojúhelníku a směr normály

Vazba trojúhelník–tetrahedron je řešena následujícím způsobem – pointer `ta1` ukazuje na tetrahedron, pro nějž je normála trojúhelníku orientována směrem **ven**. Pointer `ta2` pak nutně ukazuje na tetrahedron, pro nějž normála trojúhelníku ukazuje **dovnitř**. Pokud je hodnota některého ukazatele rovna `NULL`, tetrahedron v tomto směru neexistuje. Výše uvedené odkazy jsou kromě testů na vzájemnou pozici bodu a tetrahedronu rovněž používány při zjišťování sousednosti tetrahedronů.

3.2.3 Tetrahedrony

Každému tetrahedronu je vyhrazena struktura `tTetrahedron`, tetrahedrony jsou podobně jako trojúhleníky organizovány v obousměrně zřetězeném spojovém seznamu.

```
typedef struct sTetrahedron {
  long tn;  // element number
  struct sTetrahedron *next;
  struct sTetrahedron *prev;
  long flags;
  struct sTriangle *ta1;  // triangles
  struct sTriangle *ta2;
  struct sTriangle *ta3;
  struct sTriangle *ta4;
  struct sTetrahedron *thc1;  // children tetrahedra (1, 2, 3 and 4)
  struct sTetrahedron *thc2;
  struct sTetrahedron *thc3;
  struct sTetrahedron *thc4;
} tTetrahedron;
```

Každý tetrahedron je složen ze čtyř trojúhelníků, proto základem jsou odkazy na čtyři trojúhelníky. Pro testování vzájemné polohy bodu a tetrahedronu nejsou tyto odkazy dostačující, neboť neznáme směry normálových vektorů. Za tímto účelem byly zavedeny příznaky pro rychlé rozpoznání směru normály trojúhelníka `TETRAHEDRON_REV_TA1`, ...`REV_TA2`, ...`REV_TA3` a ...`REV_TA4`. Jsou-li nastaveny, označují normálu ukazující dovnitř tetrahedronu, jinak má normála směr vně tetrahedronu.

Speciální odkazy řešící sousednost tetrahedronů nejsou nutné, použít lze odkazy z trojúhelníků. Ukládání odkazů na vrcholy rovněž není nutné, neboť jsou snadno zjistitelné z libovolných dvou trojúhleníků – tři vrcholy získáme z jednoho a čtvrtý jako odlišný vrchol z druhého.

V návrhu struktury pro tetrahedron je rovněž předpokládáno použití DAG struktury. Tetrahedron může být dělen na čtyři, tři nebo dva nové a optimalizován na tři nebo dva nové – nejvíce tedy na čtyři potomky. Proto odkazy `thc1` až `thc4` dostačují. Tyto odkazy musí být v souladu s dříve uvedenými odkazy na trojúhleníky – tetrahedron-potomek se musí nacházet na stejné pozici jako jeho bázový trojúhelník (ten, který posloužil jako základna pro jeho konstrukci). Zpětné odkazy na rodiče nejsou nutné, neboť struktura je používána jednosměrně.

3.3 Klíčové části programu

Tato část detailně popisuje hlavní manipulace s datovými strukturami při jednotlivých případech dělení tetrahedronu a při lokálních transformacích. Z jednotného popisu celého programu, který je uveden dále, je tato část vyjmuta záměrně, neboť je obsáhlejší a narušovala by tak stručnost celkové koncepce této části.

Během inkrementálního vkládání jsou veškeré operace se strukturami prováděny s ohledem na DAG strukturu. To znamená jednak používání odkazů na potomky a jednak označování dělených a transformovaných tetrahedronů speciálními příznaky, které posléze zjednoduší odstranění nepotřebných vazeb rodič–potomek. Tyto příznak jsou typu `TRIANGLE_DIVIDED` nebo `TRIANGLE_CHANGED` a `TETRAHEDRON_DIVIDED` nebo `TETRAHEDRON_CHANGED`, jejichž názvy pokračují typem provedeného dělení nebo transformace.

Během post-optimalizace, před níž je DAG struktura odstraněna neboť již není zapotřebí, jsou lokální transformace prováděny přímo na aktuální síti bez použití jakýchkoliv speciálních vazeb.

3.3.1 Dělení tetrahedronů

První z důležitých akcí nad tetrahedronovou sítí je dělení tetrahedronu během inkrementálního algoritmu. Jak bylo uvedeno dříve, mohou nastat tři různé případy dělení tetrahedronu: vytvořeny jsou buď čtyři nové, tři nové (šest nových) a neznámý počet nových tetrahedronů. V implementaci řeší každý tento případ speciální funkce, která vytváří nové elementy a upravuje vazby mezi nimi a původní sítí.

Funkce `Make4Tetrahedra()` Řeší případ, kdy nově přidaný bod leží uvnitř tetrahedronu. Parametry této funkce jsou ukazatel na dělený tetrahedron `pth` a index přidávaného bodu `v5`. Funkce vrací boolovskou proměnnou, jejíž hodnota informuje o úspěšnosti či neúspěšnosti celé akce.

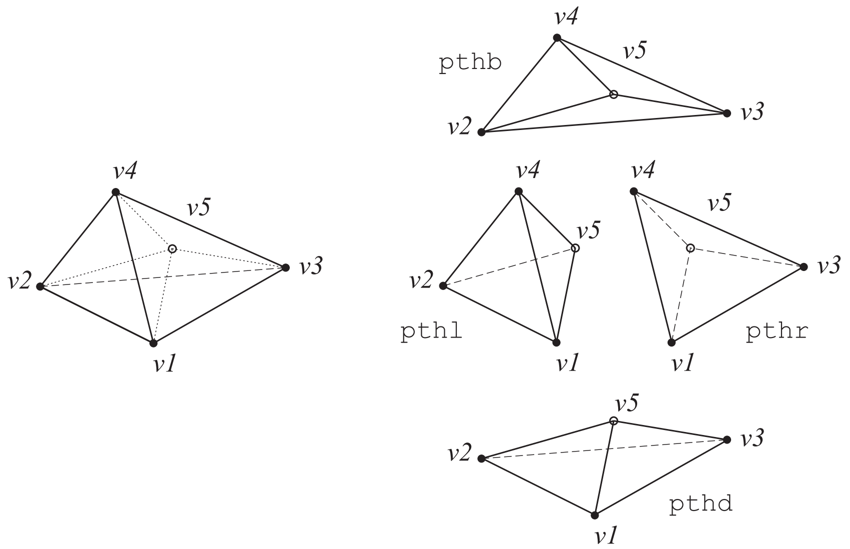
Nejprve probíhá alokace struktur pro všechny nové prvky sítě. Poté je provedena inicializace nově vytvořených trojúhelníků – ta je však závislá na vazbě trojúhelníků a bodů původního tetrahedronu. Vrcholy tetrahedronu jsou totiž načítány ze struktur trojúhelníků a pro správnou orientaci nových trojúhleníků je zapotřebí znát orientaci starých trojúhelníků, resp. prvního z nich. Pokud je orientace jeho vrcholů „očekávaná“ – normála míří ven z tetrahedronu – je vše v pořádku. Pokud má orientace opačný smysl, je třeba pro správnou konstrukci provést změnu pořadí právě získaných vrcholů.

Dalším krokem je zjištění konfigurace trojúhelníků původního tetrahedronu, které tvoří **bázové** trojúhelníky nových tetrahedronů. Značení `ptad`, `ptal`, `ptar`, `ptab` vyplývá z jejich pozice – dolní, levý, pravý a zadní. Obdobná jména jsou užita u nových tetrahedronů – viz obr 3.3. Současně s tímto procesem probíhá napojení nových tetrahedronů na rodiče v souladu s pravidlem o bázových trojúhelníkách.

Nakonec jsou doplněny všechny odkazy nutné pro konzistenci celé struktury: novým tetrahedronům jsou doplněny odkazy na všechny jejich trojúhelníky a podle bázových trojúhelníků rodiče jsou nastaveny případné příznaky pro obrácené směry normál. U bázových trojúhelníků jsou aktualizovány odkazy na nově vzniklé tetrahedrony, naplněny jsou rovněž odkazy nových trojúhleníků na nové tetrahedrony.

Funkce `Make6Tetrahedra()` Další případ dělení nastává, pokud přidávaný bod leží na stěně tetrahedronu. Parametry funkce jsou ukazatel na tetrahedron `pth`, index přidávaného bodu `v5` a pozice děleného trojúhelníku `pos`. Návratová boolovská proměnná informuje o úspěšnosti provedené akce.

V případě bodu ležícího na stěně tetrahedronu, může být rozdělen jeden nebo dva tetrahedrony. Záleží na tom, zdali se trojúhelník nachází v konvexní obálce či



Obrázek 3.3: Detailní nákres dělení, pokud nový bod leží uvnitř tetrahedronu

nikoliv. Rozdělení každého tetrahedronu může být provedeno do samostatně podobným postupem, proto je funkce `Make6Tetrahedra()` logicky dekomponována na dvě volání funkce `Make3Tetrahedra()` (druhé volání se provede pouze v případě, je-li ho zapotřebí).

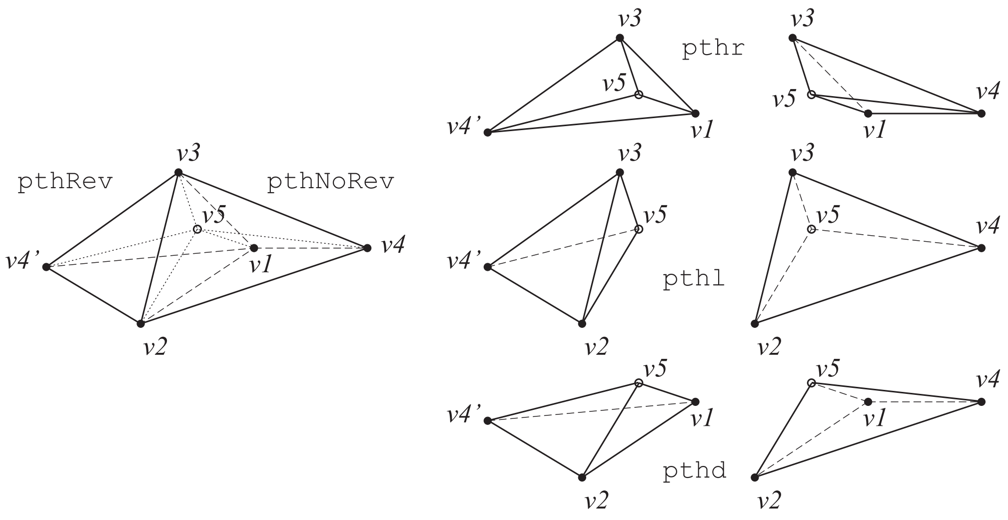
Ještě před tím, ještě ve společné části, je vybrán ukazatel na dělený trojúhleník, a tento je následně nahrazen třemi novými. Jsou také vybrány ukazatele na oba tetrahedrony, které pak slouží jako parametr `pth` při volání funkce `Make3Tetrahedra()`. Dalšími parametry jsou dělený trojúhelník `ptaDivided`, index vkládaného bodu `v5`, nově vytvořené trojúhelníky `ptad`, `ptal`, `ptar` a boolovský příznak `bDividedIsRev` udávající orientaci děleného trojúhelníku – neboli který z tetrahedronů po obou jeho stranách je zpracováván.

Funce `Make3Tetrahedra()` pak provádí vlastní alokaci nových prvků, tedy tří nových tetrahedronů a tří vnitřních trojúhelníků, jejichž inicializace závisí na příznaku obráceného směru normály. Podobně jako v při předchozím dělení je nutno zjistit rozmístění ostatních trojúhelníků v rodičovském tetrahedronu, které slouží jako bázové pro nové tetrahedrony. Jejich značení je opět závislé na jejich umístění – `ptadth` dolní, `ptalth` levý, `ptarth` pravý. Také během této činnosti probíhá vytváření vazeb z rodičů na potomky.

V závěru jsou opět doplněny odkazy mezi starými a novými elementy tak, aby celá síť byla v konzistentním stavu.

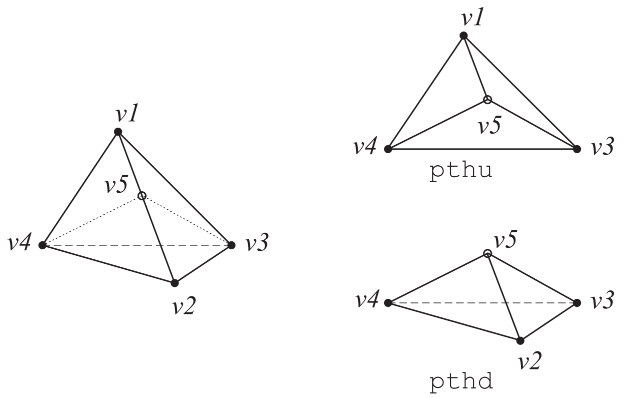
Funkce `MakeNTetrahedra()` Leží-li vkládaný bod na hraně tetrahedronu, jedná se o poslední možný případ dělení. Funkce realizující tento typ dělení se jmenuje `MakeNTetrahedra`. Jejími parametry jsou ukazatel na jeden z tetrahedronů sdílejících hranu `pthFirst`, přidávaný bod `v5` a pozice dvou stěn (trojúhelníků) `pos1` a `pos2`, které sdílejí zmiňovanou hranu. Návratová hodnota je tradičně předávána v boolovské proměnné informující o úspěšnosti provedené akce.

Funkce nejprve získá odkazy na dělené trojúhelníky, pomocí nichž nalezne indexy vrcholů rozdělené hrany. Poté jsou její indexy prohozeny tak, aby orientace nově vytvářených trojúhelníků a tetrahedronů byla vždy stejná. Následně je první „boční“ trojúhelník rozdělen. Následné rozdělení všech čtyřstěnů kolem hrany je provedeno v cyklu.



Obrázek 3.4: Detailní nákres dělení, pokud nový bod leží na stěně tetrahedronu

Jsou alokovány a inicializovány tři nové struktury pro trojúhelníky (dva boční a jeden uvnitř) a dvě nové struktury pro tetrahedrony. Na vyhledáných dolním a horním bázovém trojúhelníku `ptad` a `ptau`, se vytvoří nové tetrahedrony. Doplněním zbylých odkazů opět dostaneme síť do konzistentního stavu. Nalezením dalšího souseda ve směru pohybu kolem hrany se celý proces v cyklu opakuje pro všechny tetrahedrony sdílející hranu. Cyklus je ukončen, když se po obvodu celé hrany dostaneme opět k počátečnímu tetrahedronu.



Obrázek 3.5: Detailní nákres dělení, pokud nový bod leží na sdílené hraně

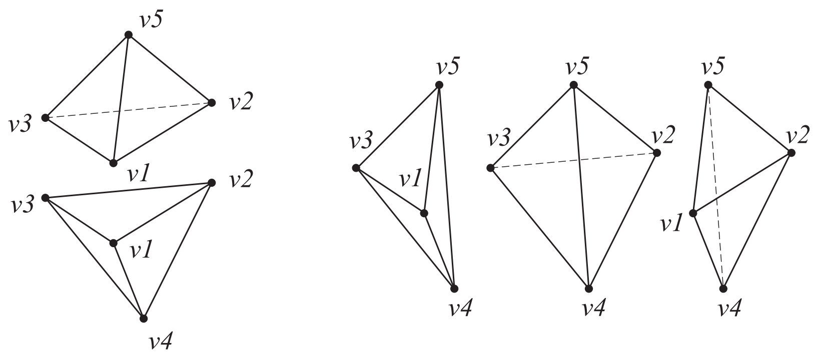
3.3.2 Lokální transformace

Druhou klíčovou akcí v algoritmu jsou lokální transformace. Z implementačního hlediska je pro každou transformaci vytvořena speciální funkce, která provádí tuto elementární operaci. Společným parametrem předávaným do všech funkcí je typ prováděné transformace `eTOT`. Tento parametr rozlišuje, má-li být transformace provedena v prostředí DAG struktury (hodnota `TOT_DAG_OPTIMIZE`) či zdali se pracuje nad „čistou“ sítí (hodnota `TOT_HARD_OPTIMIZE`). V programu se v závislosti na tomto parametru provede napojení nových elementů buď do vyhledávací struktury, nebo přímo na prvky sítě.

Na konci všech transformačních funkcí jsou vkládány trojúhelníky konvexní obálky do optimalizační fronty. Tato činnost vyplývá z podstaty optimalizace, vysvětlena byla dříve v teoretické části.

Funkce `ChangeCfg1aToCfg1b()` Jediným předávaným parametrem je ukazatel na optimalizovaný trojúhleník `pta`. Funkce tak může jednoduše zjistit oba sousedící tetrahedrony. Po alokaci a inicializaci tří nových struktur pro trojúhelníky a alokaci tří nových struktur pro tetrahedrony jsou dohledávány ukazatele na trojúhelníky konvexní obálky, aby mohlo dojít k provázání původní a nově vytvořené části.

Obrázek 3.6 vlevo ukazuje tetrahedrony v konfiguraci 1a, zatímco vpravo je stejná konfigurace pěti bodů po transformaci, tedy v konfiguraci 1b. Na obr. 3.7 vpravo je kvůli snazší orientaci v názvech proměnných uveden schematický náčrt pojmenovávání nově vytvářených tetrahedronů. Tučně bezpatkově jsou značeny tetrahedrony, bezpatkovým skloněným písmem trojúhelníky. Vrcholům je ponecháno jejich původní značení.



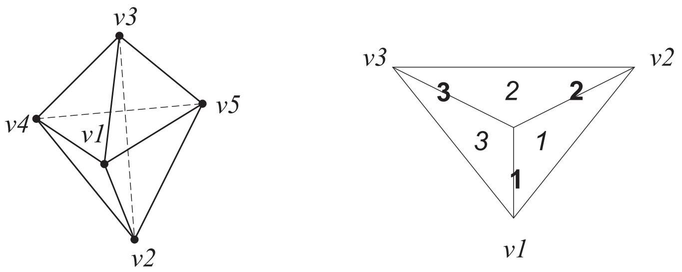
Obrázek 3.6: Detailní nákres změny konfigurace z 1a na 1b

Funkce `ChangeCfg1bToCfg1a()` Jako hlavní parametr je zde předán ukazatel na trojúhelník `pta1`. V konfiguraci 1b je nutno potvrdit nebo vyvrátit existenci třetího tetrahedronu vyplňujícího konvexní obálku. Tuto činnost řeší již před tím funkce pro zjištění konfigurace, proto jsou jako další parametry předány následující údaje – ukazatele na zbylé dva vnitřní trojúhelníky `pta2` a `pta3`, ukazatel na dohledaný tetrahedron `pth2` a vrcholy `v1`, `v2` a `v3` ve správném pořadí. Vytvářen je pouze jeden nový trojúhelník a dva nové tetrahedrony, dohledávání a doplňování odkazů probíhá obdobně jako v předchozí funkci.

Obr. 3.7 vpravo předkládá označení vrcholů v této konfiguraci, optimalizovaným trojúhelníkem je zde $v1v2v3$. Nákres celkového rozložení elementů je na tomtéž obrázku vpravo.

Funkce `ChangeCfg3()` Oproti předchozím dvěma transformacím, kdy na každou bylo nutno použít jednu funkci, je transformace typu 3 jednodušší v tom smyslu, že na její provedení stačí funkce jedna. To je způsobeno tím, že obě konfigurace mají stejný počet elementů, a liší se pouze ve značení. Detailní nákres transformace je předložen na obr. 3.8 v levé části.

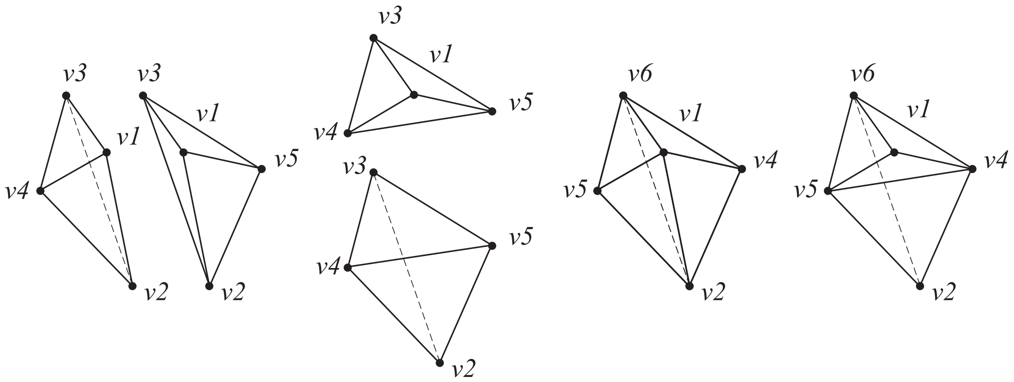
Tato konfigurace se od předchozích liší ještě jedním aspektem. Její transformovatelnost je závislá na uspořádání sousedních tetrahedronů, konkrétně přes troj-



Obrázek 3.7: Vlevo nákres značení vrcholů při transformaci z konfigurace 1b na 1a, vpravo naznačeno pojmenování vrcholů a tetrahedronů při pohledu shora

úhelníky $v1v2v4$ a $v2v1v5$. Pokud tyto sousední tetrahedrony nejsou sobě navzájem sousedy, tzn. neexistuje jejich společný vrchol $v6$, není tato sousední konfigurace transformovatelná, čímž se stává netransformovatelou i původní konfigurace. V opačném případě mohou být provedeny obě poloviny transformace, druhá část je znázorněna v levé části obr. 3.8.

Předávané parametry jsou kromě ukazatele na optimalizovaný trojúhelník `pta1` také ukazatele na oba komplanární trojúhelníky `ptal` a `ptar`, oba tetrahedrony `pthl2` a `pthr2`, orientované vrcholy `v1`, `v2`, `v3` a společný vrchol sousedních trojúhelníků `v6`. Problém je dekomponován na funkci `ChangeCfg3Half()` zpracovávající polovinu transformace. Dvojitým použitím této funkce je provedena vlastní transformace. Předtím je ještě zpracována společná část rozdělení komplanárních trojúhleníků.

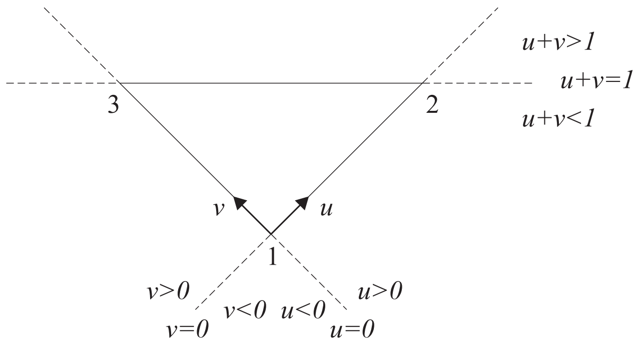


Obrázek 3.8: Detailní nákres změny konfigurace 3, vpravo ukázka značení na sousedních tetrahedronech před a po transformaci

3.3.3 Určení typu konfigurace

Před provedením vlastní transformace při optimalizaci trojúhleníku musíme zjistit, v jaké konfiguraci vůči svému okolí se tento trojúhelník nalézá. Po jejím určení můžeme provést patřičné kroky, například lokální transformaci. Určení konfigurace je provedeno na základě výsledku testu na průsečík přímky s trojúhelníkem. Je-li trojúhelník sdílen dvěma tetrahedrony, čtvrté vrcholy těchto tetrahedronů poslouží jako body určující přímku. Možné situace jsou na obr. 3.9.

Určení průsečíku je prvním krokem, v některých případech je pro určení kon-



Obrázek 3.9: Určení vzájemné polohy přímky a trojúhelníku

figurace nutno podniknou další akce. Následuje výčet možných konfigurací spolu s jejich stručným popisem:

`VC_OTHER` tato konfigurace nastane v případě účasti trojúhelníku v konvexní obálce, tedy pokud neexistují oba tetrahedrony a není možno přikročit k testu průsečíkem; dalším možným případem je selhání nějakého přídavného testu v dalších typech konfigurací,

`VC_5` čtyři body jsou komplanární, tzn. přímka protíná některou z přímek, na níž leží hrana trojúhelníku; hrana však protnuta není,

`VC_4` čtyři body jsou komplanární, přímka prochází některým vrcholem trojúhelníku,

`VC_3` čtyři body jsou komplanární, přímka protíná některou z hran trojúhelníku; v této konfiguraci je dále nutno dohledat dva sousední tetrahedrony a otestovat, zda mají společný vrchol – pokud ne, je konfigurace označna za `VC_OTHER`,

`VC_1a` průsečík se nachází uvnitř trojúhelníku,

`VC_1b` průsečík se nachází mimo trojúhelník, ale v oblasti výseče, která dává naději na nalezení této konfigurace; dále je ještě nutno dohledat tetrahedron zaplňující konvexní obálku – je to ten, jež je sousedem oběma sdíleným tetrahedronům, pokud neexistuje, je konfigurace prohlášena za `VC_OTHER`,

`VC_2` vzájemná poloha trojúhelníku a přímky nebyla určena pro žádnou jinou předchozí konfiguraci, průsečík leží mimo trojúhelník ve špatné výseči (bod leží uvnitř tetrahedronu).

Parametrem funkce je ukazatel na trojúhleník, jehož konfiguraci zjišťujeme. Funkce dále naplňuje konfigurační strukturu údaji, které zjistila např. při dohledávání chybějících tetrahedronů, a které tak nebudou muset být znovu zjišťovány transformačními funkcemi. Návratovou hodnotou je typ konfigurace, nebo speciální typ konfigurace `VC_FAIL`, signalizující selhání během výpočtu.

3.4 Přehledově o programu

Protože se jedná o implementačně značně rozsáhlý problém, je tento dekomponován na podproblémy, které jsou – jak je v jazyce C zvykem – implementovány v několika souborech. Následuje jejich výčet, u každého souboru je uveden stručný popis. Tento si nedělá nárok ani na dokonalost ani na úplnost. Je spíše přehledový, neboť na přiloženém CD jsou k dispozici kompletní zdrojové soubory:

`geom.c` funkce implementující části související s analytickou geometrií – práci se základními geometrickými elementy, jakými jsou vektory a roviny; výpočet koule vepsané o opsané; výpočty hodnot tvarových kritérií, jež hodnotí kvalitu tetrahedronu; zde jsou implementovány základní funkce pro výpočty s tolerancí,

`inct3d.c` implementace inkrementálního algoritmu: zde se nachází procedury pro lokaci bodu uvnitř tetrahedronu a pro dělení tetrahedronu; důležitou částí je též odstranění hraničního tetrahedronu,

`initerm.c` tento soubor sloužil z větší části pouze během ladění programu, některé jeho části byly buď přemístěny do jiných souborů, neboť sloužily při ladění jako náhražka okolních modulů; koncentruje v sobě ladicí výpisy, a funkce pro inicializaci a ukončování celého triangularizačního procesu, generátor bodů, apod.,

`lists.c` obsahuje funkce pro práci s obousměrně zřetězenými seznamy trojúhelníků a tetrahedronů

`optim.c` zahrnuje funkce pro zjištění konfigurace tetrahedronů a pro lokální transformace, jež jsou dále využity při optimalizačním a post-optimalizačním procesu; dále obsahuje jednotné rozhraní pro výpočet hodnot tvarových kritérií,

`t3d.c` v ladící fázi programu se jednalo o hlavní část obsahující funkci `main()`; zde byl celý proces tetrahedronizace spouštěn, zde bylo simulováno okolí,

`macros.h` tento soubor obsahuje definice symbolických konstant a maker společně užívaných v celém programu; lokální konstanty a makra jsou definovány pouze v těch hlavičkových souborech, kde jsou zapotřebí,

`ttypes.h` definice hlavních datových typů užívaných programem – jedná se zejména o struktury hlavních geometrických elementů a výčtové typy.

3.5 Začlenění do MVE

3.5.1 Co je MVE?

MVE je zkratka pro **Modular Visualization Environment**. Jedná se o prostředí, které má uživatelům poskytnout jednoduchou obsluhu složitých výpočetních modulů, programátorům pak možnost jednoduchého zabudování vlastních částí mezi již existující celky.

Uživatelům je dán k dispozici editor, který umožňuje interaktivní spojování a spouštění různých modulů. Modulem se zde rozumí „obyčejný“ program, který produkuje nějakou činnost – například generuje či nahrává data ze souboru, nebo je naopak ukládá. Hlavní skupinu však tvoří moduly výpočetní, které nad zadanými daty provádí „zásadnější“ operace – nejčastěji se jedná o implementace různých algoritmů. Tento přístup s sebou přináší výhody spojené s možností využití již napsaných modulů. Např. plně postačuje mít jeden vizualizační modul, který zobrazuje výsledky ostatních výpočetních modulů – můžeme tak odstranit zbytečné programování podobných vizualizačních částí ve více různých aplikacích.

Z programátorského hlediska je MVE řešeno jako systém dynamických knihoven pod operačním systémem WindowsNT. V jedné knihovně může být umístěno i několik modulů, přičemž každý modul je vytvořen pro svoji konkrétní činnost. Knihovny komunikují s editorem pomocí jednoduchého rozhraní několika jasně definovaných funkcí.

Moduly většinou slouží ke zpracování dat, proto je nutné jim předávat vstupy a odebírat výstupy. Ty jsou mezi moduly vyměňovány pomocí jednotných datových struktur, přičemž každý z modulů dává uživateli najevo, která data je schopen zpracovávat.

3.5.2 Přizpůsobení programu pod MVE

Program, tak jak byl uveden dříve, byl vytvořen jako klasický `EXE` soubor. Pro začlenění pod MVE bylo nutno pouze přeprogramovat část pro komunikaci s okolím, tedy načtení vstupů a produkci výsledků a umístit celý program do `DLL` knihovny. K tomuto účelu byly přidány další zdrojové soubory:

`DLLInterface.c` vytvářející část pro komunikaci jednotlivých modulů s okolím; zde jsou umístěny funkce exportované vně knihovny a funkce identifikující moduly v knihovně se nacházející,

`DLLMisc.c` zde jsou umístěny některé funkce přemístěné sem ze zrušených souborů z dřívějších verzí programu v jeho `EXE` podobě, zejména části sloužící pro náhodné generování bodů,

`tetra.cpp` hlavní soubor celého projektu,

`iso.c` tělo modulu napsaného za účelem vizuálního hodnocení vygenerovaných sítí; tento modul implementuje jednoduchý algoritmus generování izoploch nad danou tetrahedronizací,

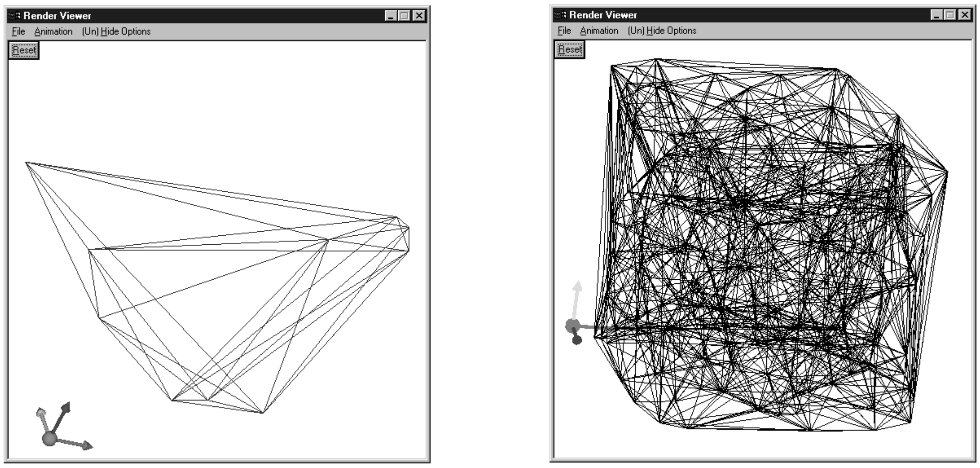
`Unit2.cpp` programová jednotka vytvářející uživatelské rozhraní pro modul generující tetrahedronovou síť,

`UnitPointGen.cpp` programová jednotka dialogového okna generátoru bodů,

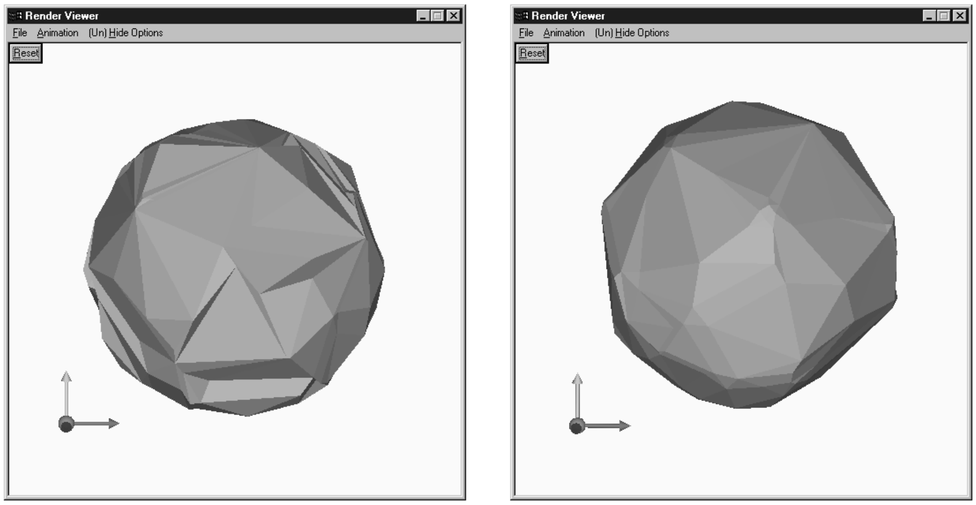
`UnitTetraHist.cpp` jednoduchý zobrazovač histogramů různých tvarových kritérií.

V dynamické knihovně byly kromě základního modulu pro generování tetrahedronových sítí implementovány ještě další moduly. Přehledově se jedná o generátor vstupních bodů, generátor izoploch nad tetrahedronovou sítí, převodník tetrahedronové sítě na trojúhelníkovou a jednoduchý zobrazovač histogramů tvarových kritérií. Blíže jsou moduly a jejich uživatelské rozhraní popsány v přílohách.

Jako ukázka sítí vytvořených pod MVE je začleněn obr. 3.10, který ukazuje okno vizualizačního modulu s vygenerovanou tetrahedronovou sítí pro množinu deseti resp. tří set bodů. Další ukázkou jsou vygenerované izoplochy na obr. 3.11. Každá byla vytvořena nad množinou 500 bodů a post-optimalizována vůči jinému tvarovému kritériu. Je zřejmé, že tvary tetrahedronů rozhodují o kvalitě vizuálního vjemu při rekonstrukci povrchu.



Obrázek 3.10: Ukázka vytvořených sítí – vlevo síť s 10 body, vpravo síť se 300 body



Obrázek 3.11: Ukázka vytvořených izoploch pro různé sítě s 500 body – vlevo pro kritérium MaxVolume, vpravo pro MaxMinSolidAngle

Kapitola 4

Výsledky

Algoritmus popsaný v předchozí části byl implementován v programovacím prostředí Borland C++ Builder 3.0 pod Windows NT. Testování proběhlo na PC PII 350 MHz, 64 MB RAM.

Pro testování byly generovány náhodné datové množiny s několika typy rozložení pravděpodobnosti:

rovnoměrná data jsou v prostoru rozmístěna nezávisle,

gaussovská data mají zadaný rozptyl kolem střední hodnoty – kolem jednoho bodu v prostoru je vytvořen shluk bodů, s rostoucí vzdáleností je výskyt bodů řidší,

clustery – pět rovnoměrně náhodně rozmístěných hodnot vytvoří středy pro clustery, každý z clusterů má Gaussovo rozložení pravděpodobnosti,

přímky – v prostoru je rozmístěno deset bodů, vždy dva z nich tvoří koncové body úsečky; na těchto úsečkách jsou náhodně s rovnoměrným rozložením pravděpodobnosti generovány vstupní body.

Časové poměry výpočtu mezi jednotlivými rozloženími bodů udávají tabulky 4.1 až 4.4, jejichž hodnoty jsou vyneseny do grafu 4.1. V tabulce 4.1 jsou časové hodnoty získané pro pět různě velkých datových množin, ke získání průměrné hodnoty bylo provedeno pět měření.

Počet bodů	500	1000	1500	2000	2500	3000	3500	4000
Čas [ms]	851	1762	2714	3725	4646	5788	6790	7861
Čas [ms]	841	1783	2714	3735	4747	5798	6719	8452
Čas [ms]	821	1793	2744	3746	4697	5718	6739	7761
Čas [ms]	821	1782	2794	3675	4727	5748	6670	7741
Čas [ms]	821	1772	2744	3726	4707	5788	6690	7811
Průměrný čas [ms]	831	1778	2742	3721	4705	5768	6722	7925

Tabulka 4.1: Časy naměřené pro rovnoměrné rozložení bodů

Časy naměřené pro gaussovské rozložení bodů jsou uvedeny v tabulce 4.2, pro shluky v tabulce 4.3 a pro rozložení bodů na přímkách v tabulce 4.4.

Vynesením naměřených hodnot do grafu získáme obrázek 4.1, jež ilustruje časové závislosti pro různá rozdělení bodů.

S velikostí paměti 64 MB bylo možno testovat datové množiny čítající do 5000 bodů bez nutnosti odkládání na disk. U rozložení bodů na přímkách se tato hranice pohybovala kolem 700 bodů.

Počet bodů	500	1000	1500	2000	2500	3000	3500	4000
Čas [ms]	842	1823	2864	3775	4837	5899	6910	8322
Čas [ms]	861	1833	2804	3805	4827	5848	6870	7902
Čas [ms]	841	1812	2804	3785	4867	5899	6920	7881
Čas [ms]	852	1833	2804	3826	4817	5839	6940	7901
Čas [ms]	851	1813	2824	3816	4857	5879	6940	8061
Průměrný čas [ms]	849	2820	2742	3801	4841	5873	6916	8013

Tabulka 4.2: Časy naměřené pro gaussovské rozložení bodů

Počet bodů	500	1000	1500	2000	2500	3000	3500	4000
Čas [ms]	781	1732	2734	3655	4707	5638	6550	8212
Čas [ms]	781	1732	2724	3615	4576	5598	6439	7701
Čas [ms]	781	1692	2684	3685	4696	5568	6489	7651
Čas [ms]	791	1682	2613	3795	4517	5528	6649	7621
Čas [ms]	802	1733	2644	3675	4687	5648	6700	7551
Průměrný čas [ms]	787	1714	2680	3685	4637	5596	6565	7747

Tabulka 4.3: Časy naměřené pro rozložení bodů ve shlucích

Jako testovací množiny pro zjišťování vlastností Delaunayových čtyřstěnů byly vybrány všechny výše uvedené možnosti rozložení bodů v prostoru. Každé z pěti tvarových kritérií bylo použito na množinu o velikosti 500, 1000 a 2000 bodů. V těchto množinách (charakterizovaných rozložením bodů, počtem bodů a počítaným kritériem) pak bylo provedeno pět experimentů, aby byl zachycen trend křivek a vyloučen vliv náhodných odchylek.

Z veškerého dalšího testování bylo vyloučeno rozložení bodů na přímkách, neboť se ukázalo jako nevhodné, díky své minimální vypovídací hodnotě a neměnným grafům.

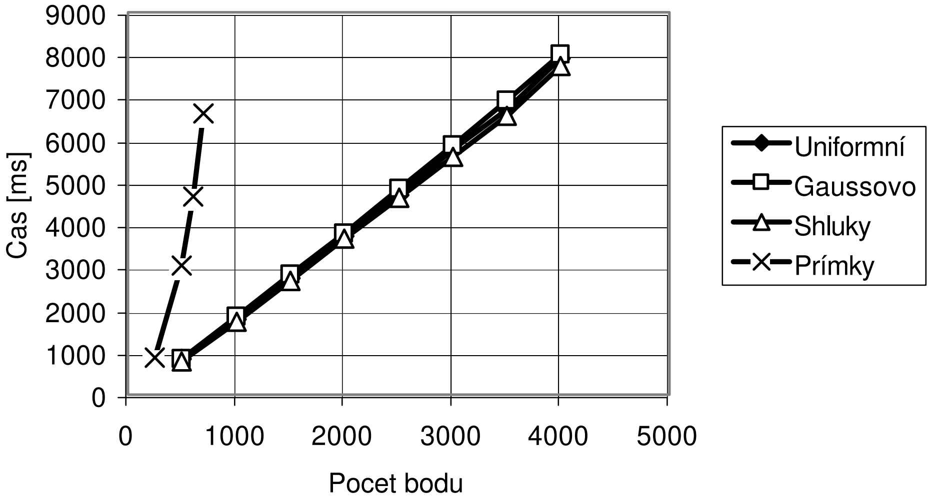
Závislost tvarů histogramů jednotlivých kritérií na velikosti vstupní datové množiny bylo prováděno na množinách o velikosti 500 až 5000 bodů s krokem po 500. Testována byla všechna kritéria na uniformním a gaussovském rozložení a na množině tvořené shluky.

Post-optimalizační testy proběhly rovněž na množinách bez použití rozmístění bodů na přímkách. Každý z jedenácti post-optimalizačních postupů byl aplikován na množiny o velikost 500, 1000, 2000 a 3000 bodů.

Výsledky byly vyneseny do histogramů, které jasně informují o rozložení kvality tetrahedronů v dané síti. V další části budou postupně představeny dosažené

Počet bodů	250	500	600	700
Čas [ms]	881	3175	5218	7731
Čas [ms]	912	2844	4286	8663
Čas [ms]	811	3155	4066	6039
Čas [ms]	971	2914	5357	5408
Čas [ms]	832	3135	4436	5318
Průměrný čas [ms]	881	3045	4673	6632

Tabulka 4.4: Časy naměřené pro rozložení bodů na přímkách



Obrázek 4.1: Časové závislosti pro různé vstupní množiny dat

výsledky ze všech částí provedeného testování.

4.1 Vlastnosti Delaunayových čtyřstěnů

Prvním výsledkem inkremetnálního algoritmu je Delaunayova tetrahedronizace. Tato síť, u které předpokládáme dobře tvarované tetrahedrony, posloužila jako první množina na otestování tvarových kritérií. Výsledky získané měřením byly zpracovány jako histogramy a vyneseny do grafů.

Titulky grafů nesou informaci o pravděpodobnostním rozložení bodů, o počtu bodů, nakonec je uvedeno jméno kritéria. Na ose x je vynesena kvalita sítě, tedy vypočítané hodnoty jednotlivých kritérií. Osa y nese informaci o relativním počtu tetrahedronů s určitou hodnotou kritéria.

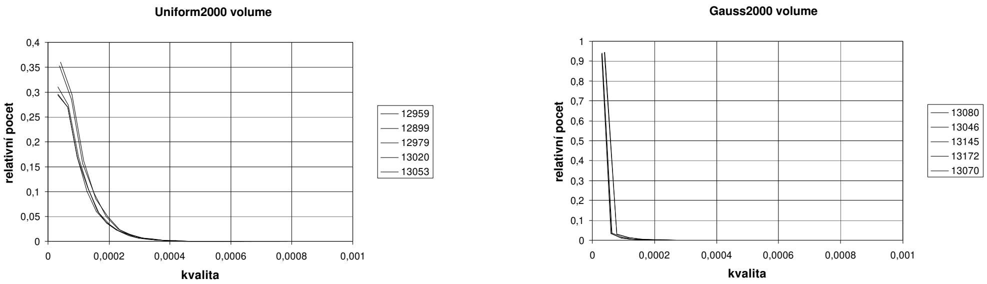
4.1.1 Objem

Většina sítí obsahuje převážně tetrahedrony s malým objemem, tetrahedronů s vyššími objemy se vyskytuje pouze zanedbatelné množství – viz obr 4.2. Tato vlastnost je téměř nezávislá na rozmístění bodů v prostoru, křivky se rozkládají pouze u jiných hodnot. Např. u Gaussova rozložení se jedná o logické posunutí směrem k nižším hodnotám, neboť u střední hodnoty jsou body více nahuštěny. Stejný efekt lze pozorovat při rostoucím počtu vstupních bodů.

Tetrahedrony s malým objemem jsou v podezření ze špatných tvarových vlastností, neboť právě ploché či špičaté tetrahedrony vykazují velmi malý objem. Naším cílem pro post-optimalizační proces je tedy zvýšit výskyt tetrahedronů s vyššími objemy.

4.1.2 Poměr koule vepsané a opsané

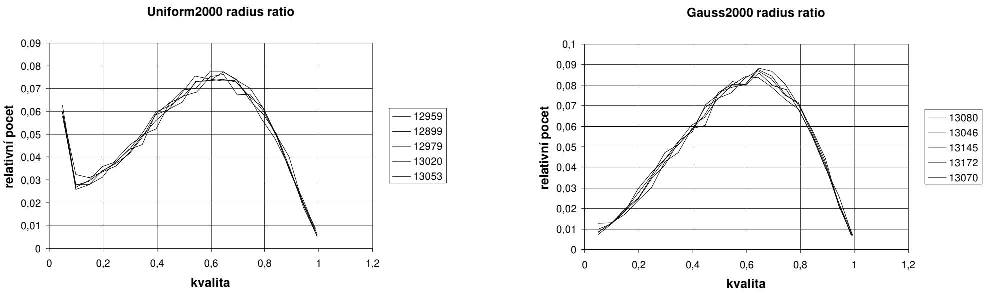
Histogram tohoto kritéria je typicky tvarovaný, viz obr. 4.3. Podobnost pro většinu vstupních množin o různém počtu bodů s různým rozložením bodů lze nalézt jak ve tvaru, tak v rozložení hodnot. Svůj vzhled postrádá pouze při umístění bodů na přímky. Maximum je většinou ve stejných místech. Jeho umístění blíže k vyšším



Obrázek 4.2: Histogramy kritéria „objem“

hodnotám napovídá, že v síti je více tetrahedronů s lepšími hodnotami než těch se špatnými hodnotami.

Přesto by bylo žádoucí pomocí post-optimalizace posunout maximum grafu k ještě vyšším hodnotám. Za úspěch by mohlo být považováno i odstranění tetrahedronů s velmi malým poměrem, čímž by se křivka grafu pro rovnoměrné rozložení mohla více podobat křivce pro gaussovské rozložení.



Obrázek 4.3: Histogramy kritéria „poměr koule vepsané a opsané“

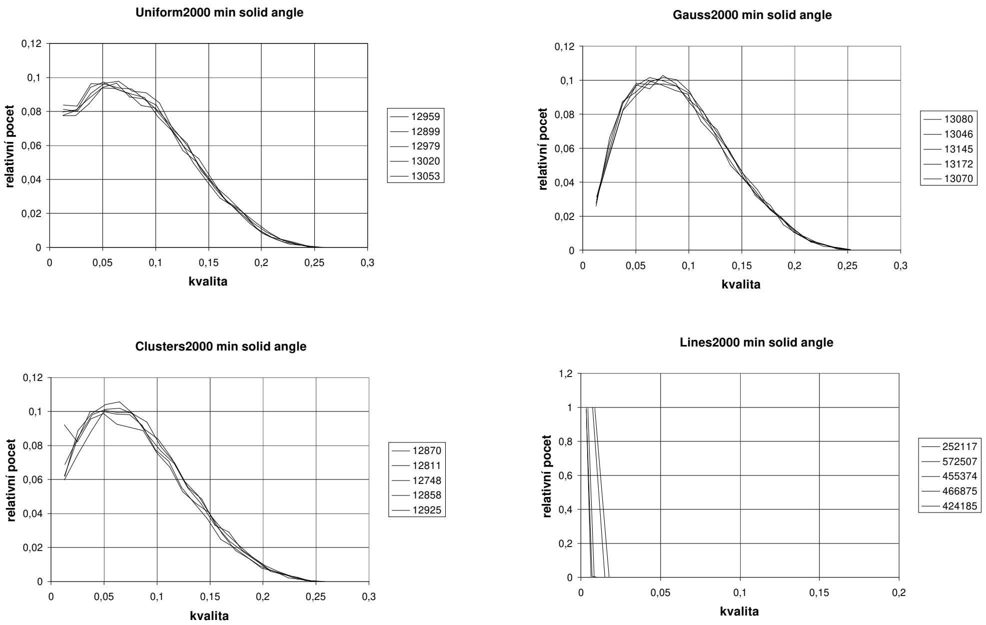
4.1.3 Plocha minimálního Eulerova trojúhelníku

Toto tvarové kritérium dosahuje – podobně jako kritérium poloměru koule vepsané a opsané – typicky tvarovaného histogramu – viz obr. 4.4. Jeho podobnost nezávisející na vstupních datech je rovněž značná, výjimku tvoří opět rozložení bodů na přímkách. Většina tetrahedronů je umístěna v nižších hodnotách, směrem k vyšším hodnotám jich ubývá. Histogram zjevně vypovídá o tom, že vytvořené tetrahedrony jsou značně vzdáleny od pravidelných tvarů rovnostranného případu.

Naším cílem pro post-optimalizaci je v tomto případě posunutí maxima křivky směrem k vyšším hodnotám. Bude-li totiž plocha minimálního Eulerova trojúhelníka větší, budou se plochy Eulerových trojúhelníků u ostatních vrcholů snižovat. Jejich vzájemné přibližování je žádoucí, neboť je totožné s vytvářením více rovnostranných tetrahedronů.

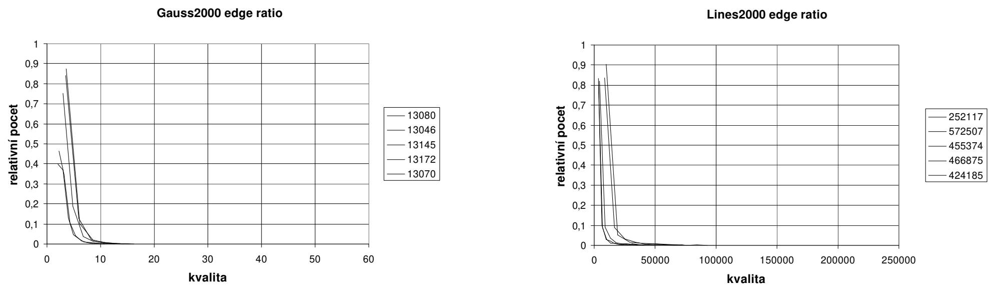
4.1.4 Poměr délek hran

Tento histogram nedává příliš užitečné informace díky značnému rozptylu hodnot, jak je vidět z obr. 4.5. Téměř nezávisle na vstupních datech je celá křivka rozložena v oblasti nízkých hodnot. Většina tetrahedronů dosahuje poměru nižšího než 10.



Obrázek 4.4: Histogramy kritéria „plocha minimálního Eulerova trojúhelníka"

Nízké poměry jsou pozitivní vlastností, i když o tetrahedronu s poměrem délek hran kolem 10 nemůžeme zdaleka prohlásit, že je rovnostranný. V post-optimalizaci by bylo vhodné pokusit se zmírnit značný rozptyl hodnot snížením maximálního poměru, nebo se pokusit posunout hlavní část grafu k ještě nižším hodnotám.

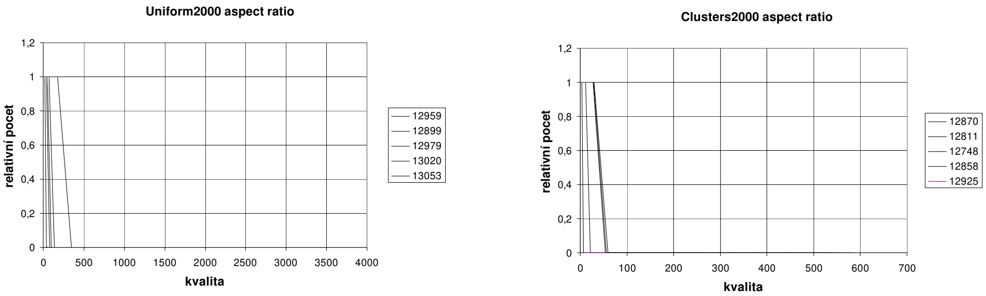


Obrázek 4.5: Histogramy kritéria „poměr délek hran"

4.1.5 Poměr poloměru koule opsané a nejdelší hrany

Rozložení hodnot je podobné histogramu poměru délek hran, pouze rozptyl hodnot je ještě vyšší, viz obr. 4.6. Vysoké maximální hodnoty naznačují značný nepoměr ve velikosti poloměru koule opsané a nejdelší hrany. Takové hodnoty jsou typické pro ploché tetrahedrony s velkým poloměrem opsané koule. Z jejich nízkého počtu se dá usuzovat na umístění v konvexní obálce.

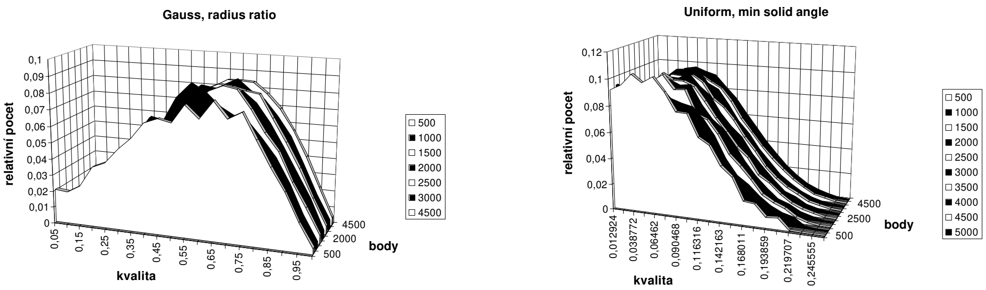
Hlavním cílem post-optimalizace v tomto případě je snížení maximálních hodnot vypočítaných poměrů. Dá se však očekávat, že tvary tetrahedronů v blízkosti konvexní obálky již nebude možno příliš vylepšit.



Obrázek 4.6: Histogramy kritéria „Poměr poloměru koule opsané a nejdelší hrany“

4.1.6 Shrnutí

Ze získaných výsledků je zjevné, že – proti původnímu očekávání – nejsou tvary tetrahedronů ve vytvářených sítích příliš závislé ani na počtu vstupních bodů (viz obr. 4.7), ani na jejich rozložení v prostoru. Výjimku tvoří rozmístění bodů na přímkách. Zde je většina histogramů podobně tvarována, nemají však příliš velkou vypovídací hodnotu. Tento jev je zřejmě způsoben počtem vytvořených tetrahedronů – pokud se jejich množství udržuje v lineární závislosti na počtu bodů, to jest v případech kdy body jsou rozmístěny „více“ náhodně, jsou si histogramy relativně podobné. Při kvadratické závislosti, tedy při větší organizovanosti bodů, dochází k vytvoření tetrahedronů podobných tvarů, které jsou narušovány pouze elementy v konvexní obálce.



Obrázek 4.7: Ukázka závislosti histogramů na počtu bodů

4.2 Výsledky post-optimalizace

V části 3.1.4 jsou uvedena kritéria, která slouží k závěrečné optimalizaci sítě. Spolu s nimi je tam uveden i způsob, kterým jsou použita. Pro přehlednost je zde uveden jejich přehledný seznam, k pojmenování jsou užity implementační názvy. Tyto budou pro stručnost používány i v dalším textu:

- `MaxRadiusRatio`
- `MaxMinRadiusRatio`
- `MaxVolume`
- `MaxMinVolume`

- `MaxMinSolidAngle`
- `MinAspectRatio`
- `MinMaxAspectRatio`
- `MinEdgeRatio`
- `MinMaxEdgeRatio`
- `MaxNumber`
- `MinNumber`

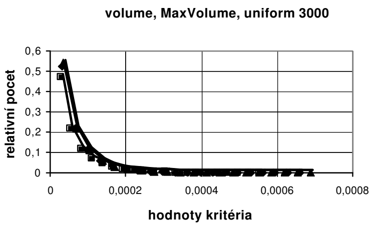
Jako jednoduchá budou dále označována kritéria typu max/min, složitějšími kritérii budou rozuměny typy maxmin/minmax.

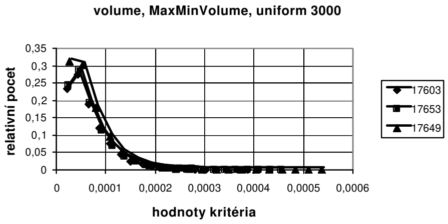
Všechny uvedené metody byly použity, výsledné sítě znovu zhodnoceny tvarovými kritérii. Přestože se dalo očekávat, že na vylepšení daného kritéria bude mít největší vliv jeho přímé použití v post-optimalizaci, i použití jiných kritérií mělo podobný výsledek. Testování proběhlo opět na datových množinách různé velikosti a různého rozložení bodů. Z procesu bylo vyloučeno pouze rozložení bodů na přímkách, neboť se v předchozích experimentech neukázalo jako vhodné.

Naším cílem je najít takové post-optimalizační kritérium, kterým by dané tvarové kritérium získalo lepší histogram, tedy větší počet „dobrých" tetrahedronů v síti. Za základní informaci o úspěšnosti či neúspěšnosti sloužilo porovnání s Delaunayovou triangulací, případně porovnání vzájemných rozdílů konečných sítí.

4.2.1 Objem

Post-optimalizací byly vytvořeny dva typy grafů, oba jsou uvedeny na obr. 4.8. Jednodušší kritéria se chovají podobně jako MaxVolume, vytvářejí hladkou křivku. Složitější pak vytváří podobně jako MaxMinVolume v levé části křivky zlom, více či méně patrný. Delaunayovo kritérium leží někde mezi, jeho zlom není příliš výrazný. Kritériem MaxVolume bylo proti jeho logice dosaženo největšího počtu tetrahedronů s malým objemem, přestože celkový počet elementů sítě nebyl v tomto případě největší.

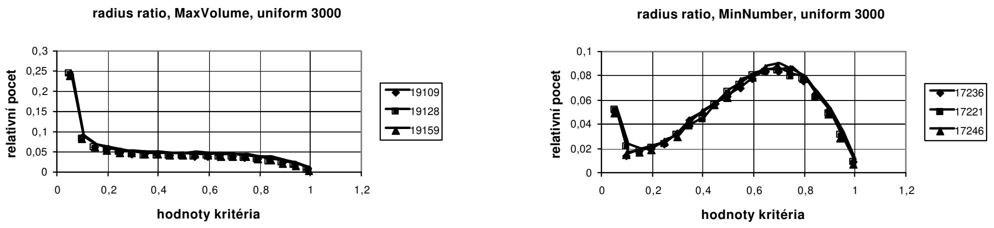




Obrázek 4.8: Post-optimalizační histogram objemu

4.2.2 Poměr koule vepsané a opsané

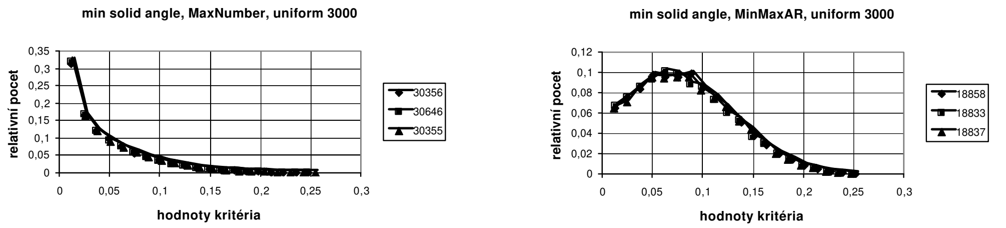
Toto kritérium vytváří tři charakteristické grafy. První je podobný známému Delaunayovskému s maximem u vyšších hodnot, na obr. 4.9 vpravo. Tento typ je obecně vytvářen složitějšími kritérii. Jednodušší kritéria naopak vytváří druhý typ grafu – většina tetrahedronů je u nižších hodnot a křivka má klesající tendenci (MaxNumber, MaxVolume, MinER), viz obr. 4.9 vlevo. Zbylá kritéria vytváří křivky podobné oběma předchozím, extrém je stále u nejnižších hodnot, ale v pravé části se rýsuje lokální maximum.



Obrázek 4.9: Post-optimalizační histogram poměru koule vepsané a opsané

4.2.3 Plocha minimálního Eulerova trojúhelníku

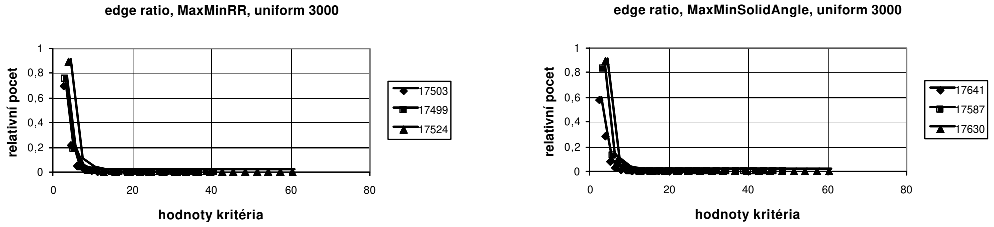
V tomto případě vznikají dva druhy grafů. První, podobný Delaunayovskému, je typycký pro složitá kritéria. Globální extrémem má u nižších hodnot, viz obr. 4.10 vlevo. Levý konec křivky je více či méně zalomen vzhůru – tento jev se zdá nepodstatný, neboť se jedná o lokální extrém. Druhý typ má prakticky většinu tetrahedronů u nižších hodnot a křivka má směrem k vyšším hodnotám klesající tendenci, viz obr. 4.10. Jednoduchá kritéria zde jednoznačně selhávají.



Obrázek 4.10: Post-optimalizační histogram plochy minimálního Eulerova trojúhelníka

4.2.4 Poměr délek hran

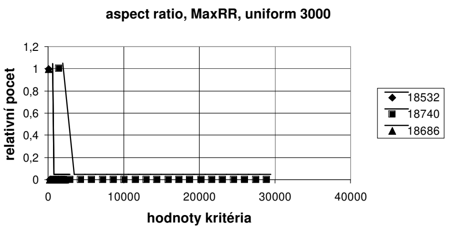
Výsledné grafy pro toto kritérium jsou v podstatě nezávislé na post-optimalizační metodě, viz obr. 4.11. Většina tetrahedronů má poměr hran menší než 10, větší poměry jsou, co do počtu vyskytujících se tetrahedronů, prakticky zanedbatelné. Maximální poměr hran se všude pohybuje kolem podobné hranice, nejnižší je – dle očekávání – u MinMaxER.

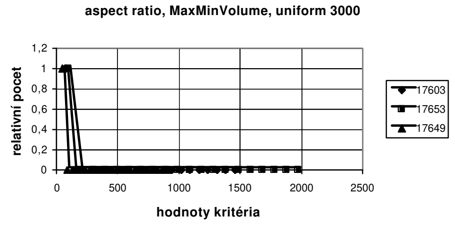


Obrázek 4.11: Post-optimalizační histogram poměru délek hran

4.2.5 Poměr poloměru koule opsané a nejdelší hrany

Toto kritérium nedává příliš přehledné výsledky, tvarově jsou si všechny grafy velmi podobné, viz obr. 4.12. Většina tetrahedronů je umístěna u nižších hodnot, počet těch s vyššími hodnotami je téměř nulový. Jednoduchá a složitá kritéria se v tomto případě odlišují maximální hodnotou počítaného poměru. Složitá kritéria mají tento poměr daleko nižší, než kritéria jednoduchá.





Obrázek 4.12: Histogramy kritéria „Poměr poloměru koule opsané a nejdelší hrany“

4.3 Zhodnocení výsledků

Implementací bylo ověřeno, že ne všechna kritéria lze použít spolu s inkrementálním algoritmem pro konstrukci tetrahedronové sítě, neboť postrádají vlastnost konvergence ke konvexní obálce původní množiny bodů. Tuto vlastnost však nepostrádá Delaunayovo kritérium. Proto mohla být 3D Delaunayova triangulace použita jako základ pro post-optimalizační proces, ve kterém byly provedeny pokusy o její další vylepšení.

Nejprve byly testovány tvarové charakteristiky tetrahedronů Delaunayovy tetrahedronizace na vstupních datech lišících se jak rozložením bodů v prostoru, tak jejich počtem. Bylo zjištěno, že histogramy tvarových kritérií tetrahedronů příliš nezávisí ani na velikosti vstupní datové množiny, ani na rozložení bodů v prostoru, pokud toto rozmístění nevede ke kvadratické závislosti a počet tetrahedronů se udržuje v mezích složitosti $\mathcal{O}(N)$. Tyto testy rovněž posloužily jako první praktické příklady chování jednotlivých tvarových kritérií.

Vytváření dalších lokálně optimálních tetrahedronizací za použití post-optimalizace vyloučilo z tvarových kritérií použitelných pro hodnocení sítí poměr délek hran. Jeho histogramy jsou téměř totožné pro veškerá vstupní data, a žádným dostupným způsobem se nepodařilo snížit vysoké hodnoty tohoto kritéria.

Poměr poloměru koule opsané a nejdelší hrany má snad ještě nižší vypovídací schopnost, neboť jeho hodnoty mají daleko větší rozptyl, přičemž většina tetrahedronů je koncentrována u nízkých hodnot. Ani zde se nepodařilo extrémní hodnoty kritérií snížit.

Ostatní kritéria se ukázala jako vhodná pro post-optimalizaci. V jednoduché variantě sice nedosahovala dobrých tvarových výsledků, tyto byly navíc spojeny se vznikem většího počtu elementů. Varianty typu maxmin/minmax však dokázaly vytvořit sítě blízké k Delaunayově tetrahedronizaci, většinou i s nepatrně menším počtem užitých tetrahedronů. K zásadnímu vylepšení Delaunayovy sítě však použití žádného kritéria nevedlo. Použití uvedených kritérií se proto může chápat jen jako alternativa k Delaunayově tetrahedronizaci, neboť ta se svými charakteristickými vlastnostmi zůstala nepřekonána.

Otázka počtu vytvořených tetrahedronů byla testována rovněž použitím post-optimalizace. Zde byla místo výpočtu tvarových kritérií použita podmínka provádění lokálních transformací zvyšujících nebo naopak snižujících počet tetrahedronů v lokální konfiguraci (transformace 1a na 1b nebo naopak). Dosažený minimální počet elementů v síti byl oproti maximálnímu téměř poloviční, tetrahedronizace s méně elementy měly daleko příznivější tvary tetrahedronů. Toto zjištění není zcela očekávané, ale o to příjemnější, neboť naším cílem je získat „dobrou“ síť s co nejmenším počtem tetrahedronů.

Kapitola 5

Závěr

Problém tetrahedronizace nad množinou rozptýlených dat je v literatuře většinou řešen konstrukcí 3D Delaunayovy triangulace. Některé z přístupů využívaných těmito algoritmy byly představeny v teoretické části.

Jako nejvhodnější metoda k implementaci byl vybrán inkrementální algoritmus s použitím lokálních transformací. Tento výběr byl učiněn jak na základě dostupných informací z literatury o robustnosti a stabilitě dostupných algoritmů, tak s ohledem na vhodnost použití lokálních transformací za účelem jednotného postupu vytváření sítí s různými vlastnostmi. Vzhledem k nekonvergenci některých kritérií k vytvoření konvexní obálky byla implementace pojata jako „vylepšování" Delaunayovy tetrahedronizace během post-optimalizačního procesu.

Implementace byla provedena pod operačním systémem Window NT 4.0, v programovacím prostředí Borland C++ Builder 3.0. Generátor tetrahedronových sítí byl začleněn jako jeden z modulů do vizualizačního systému MVE. Jako doplňující moduly byly vytvořeny generátor bodů a generátor isoploch.

Závěrečné testování vytvořených sítí bylo prováděno na množinách vstupních dat, jež se od sebe lišily jak rozložením bodů v prostoru, tak jejich počtem. Testováním Delaunayovy sítě bylo zjištěno, že histogramy vypovídající o její kvalitě jsou téměř nezávislé na vstupních množinách dat, pokud se počet vytvořených tetrahedronů udržuje v lineární závislosti na počtu bodů. Při kvadratické závislosti, kdy mnoho bodů leží na přímkách, dosahovaly výsledné hodnoty značného rozptylu a histogramy ztrácely na své vypovídací hodnotě.

Z kritérií hodnotící tvary tetrahedronů se jako nevhodná ukázala poměr délek hran a poměr poloměru koule opsané a nejdelší hrany, díky vysokému rozptylu hodnot a značné koncentraci tetrahedronů u nízkých hodnot. Ostatní kritéria byla shledána jako vhodná.

V post-optimalizačním procesu byly vyzkoušeny dva způsoby použití kritérií – jednoduché, které se snaží za každou cenu dosáhnout extrémních hodnot daného kritéria, a varianty typu maxmin/minmax, jejichž snahou je celkově vylepšit úroveň špatně tvarovaných tetrahedronů. Jednoduchý způsob nevedl k vylepšení tvarů, spíše naopak. Navíc počet vytvořených tetrahedronů byl zbytečně vysoký. Pomocí druhé varianty bylo dosaženo tvarů srovnatelných s Delaunayovým kritériem, dokonce s menším počtem elementů. K zásadnímu vylepšení Delaunayovy sítě však nedošlo.

Výsledky této práce byly nabídnuty k publikaci na Fourth International Scientific Conference on Electronic Computers and Informatics 2000, Herľany, Slovakia.

Literatura

[AEG87] Avis, D., ElGindy, H.: Triangulating Point Sets in Space. Discrete and Computational Geometry, Vol. 2, pp. 99–111, 1987.

[FP95] Fang, T.P. - Piegl, L.A.: Delaunay Triangulation in Three Dimensions. IEEE Computer Graphics and Applications, pp. 62–69, 1995.

[GD97] Golias, N. A., Dutton, R.W.: Delaunay triangulation and 3D adaptive mesh generation. Finite Elements in Analysis and Design 25, pp. 331–341, 1997.

[Joe91] Joe, B.: Construction of three-dimensional Delaunay triangulations using local transformations. Computer Aided Geometric Design, Vol. 8, pp. 123–142, 1991.

[Kol97] Kolingerová, I.: Metody triangularizace v rovině. Technická zpráva, Plzeň 1997.

[Kuř96] Kuřina, F.: Deset pohledů na geometrii. Matematický ústav AV ČR ve spolupráci s nakladatelstvím ALBRA, Praha 1996.

[LJ94] Liu, A., Joe, B.: Relationship between tetrahedron shape measures. BIT, Vol. 34, pp. 268–287, 1994.

[Rek68] Rektorys, K. a spolupracovníci: Přehled užité matematiky. SNTL, Praha 1968.

[Rou94] O'Rourke, J.: Computational Geometry in C. Cambridge University Press, 1994, New York.

[Sch93] Schumaker, L. L.: Computing optimal triangulations using simulated annealing. CAGD 10, pp. 329–345, 1993.

[SŽB96] Sochor, J., Žára, J., Beneš, B.: Algoritmy počítačové grafiky. Vydavatelství ČVUT, Praha 1996.

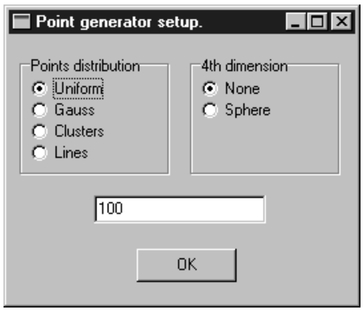
[Wat97] Watson, D. F.: Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes. Comput. J. 24, pp. 167–172, 1981.

Příloha A

Moduly začleněné do MVE

Pod systém MVE bylo kromě generátoru tetrahedronových sítí začleněno i několik dalších podpůrných modulů:

Generátor bodů `PointGen` Na obr. A.1 je zobrazeno dialogové okno nastavení. Uživatel si může zvolit mezi rovnoměrným, gaussovským rozložením bodů, zda mají být body umístěny ve shlucích nebo na přímkách. Pro generování izoploch je třeba dodat bodům dodatečnou informaci, např. o hodnotě nějaké veličny. Za tímto účelem je v nastavení umístěn panel čtvrtého rozměru, který může být buď nepoužit, nebo naplněn hodnotami, které vytváří v prostoru kouli.

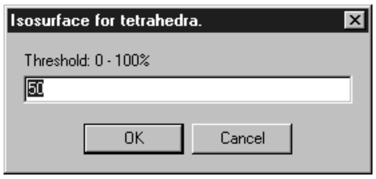


Obrázek A.1: Dialogové okno nastavení generátoru bodů

Konvertor tetrahedronové sítě na trojúhelníky `th2ta` Tento modul provede prosté „rozebrání“ tetrahedronů na trojúhelníky. Ztratí se tím sice množství užitečné informace (jak tvarové, tak např. o sousednostech), ale v době programování a testování tohoto projektu, kdy bylo zapotřebí vytvářené sítě zobrazovat, byl k dispozici pouze vizualizér trojúhelníkových sítí, jemuž se bylo nutno přizpůsobit.

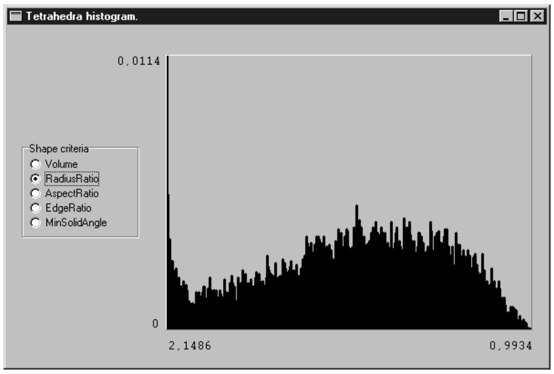
Generátor izoploch `TetraIso` Generování izoploch pracuje na principu jednoduchého algoritmu: tetrahedrony protínané izoplochou vytváří při jejím hledání buď jeden nebo dva trojúhelníky. Na obr. A.2 se nachází dialogové okno nastavení tohoto modulu, v němž si může uživatel nastavit úroveň vyhledávané izoplochy v rozmezí 0–100%.

Jednoduchý zobrazovač histogramů `TetraHist` Tento modul byl zařazen pro ukázku výsledných histogramů. Žádné nastavení parametrů není třeba, vše se ode-



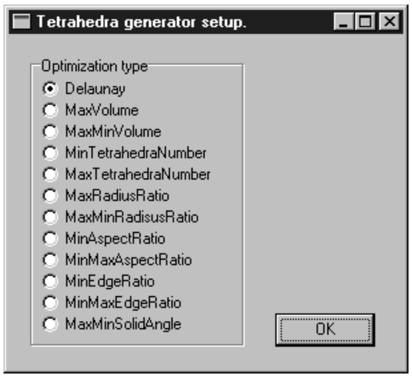
Obrázek A.2: Dialogové okno nastavení generátoru izoploch

hrává v hlavním okně programu, kde si uživatel přímo volí kritérium, jehož histogram požaduje.



Obrázek A.3: Výsledné okno zobrazeného histogramu

Generátor tetrahedronových sítí `t3d` Hlavní část celého modulu. V dialogovém okně nastavení se volí typ kritéria, na něž má být výsledná síť optimalizována, viz obr A.4.



Obrázek A.4: Dialogové okno nastavení generátoru tetrahedronové sítě

Příloha B

Instalace

Vytvořené moduly jsou umístěny v dynamické knihovně. Knihovny (moduly) jsou do prostředí MVE nahrávány prostřednictvím dialogového okna, proto je umístění knihovny na disku vcelku libovolné. Uživatel by však měl dbát doporučení a pokynů, sdělovaných mu v dokumentaci MVE.

Moduly jsou přeloženy staticky, proto by absence knihoven z programového prostředí Borland C++ Builderu v systému neměla působit závady.

Problémy se vyskytují pouze při použití formulářových oken. Borland C++ Builder vyžaduje, aby při používání formulářů byl program překládán se zarovnáním dat na 4 bytovou hranici. Protože však systém MVE vyžaduje zarovnávání dat na hranici bytu, dochází při otevírání a uzavírání formulářů k nepovoleným přístupům do paměti. Kromě modulu pro kreslení histogramů však nebyly zjištěny poruchy ve funkčnosti jednotlivých částí.

Na přiloženém CD se nalézá několik přeložených verzí knihovny, mezi nimiž je i verze se zarovnáním dat na 4 byty. Tato pracuje bez výše uvedených problémů, pouze však se stejně přeloženými moduly, tzn. nelze ji použít např. ve spolupráci s vizualizačním modulem.

Příloha C

Obsah přiloženého CD

Součástí této práce je rovněž kompaktní disk, na němž jsou uloženy důležité části vytvořené během práce na projektu. Adresářová struktura je následující:

papers v tomto adresáři by se měly nacházet případné teoretické podklady v elektronické podobě – žádné však nebyly použity,

private obsahuje zdrojové texty programu, dokumentaci ve formátu TEX a `.rtf`, obrázky ve formátu `.eps` a `.cdr` a výsledné tabulky s grafy ve formátech produktu MS Office,

public kde se nalézá výsledná dynamická knihovna v několika verzích a vysázený text diplomové práce ve formátu `.ps`.